Votre initiation à la programmation sur un système monocarte extensible

JUNIOR COMPUTER

Livre 1

PUBLITRONIC

Junior Computer 1

Votre initiation à la programmation sur un système monocarte extensible

A. Nachtmann
G.H. Nachbar

PUBLITRONIC Sarl
BP48
59930 la Chapelle d'Armentières

ISBN 2-86661-001-6
Imprimé aux Pays-Bas.

Pourquoi ce livre?

Pour beaucoup, l'ordinateur signifie moins de travail et davantage de temps libre, à d'autres il permet de tirer un meilleur rendement de leurs moments de liberté. De plus en plus d'amateurs se passionnent pour cet "esclave numérique" du XX^{o} siècle et ceux que captive cet attachant "hobby" ont le choix entre de nombreux apparails prêts à l'emploi, ou à construire soi-même. A cela s'ajoute la lecture d'ouvrages et de revues plus ou moins spécialisés dans ce domaine.
Tout cela est parfait.
Mais s'il nous faut admettre que la recherche de la solution la meilleure est menée avec tout le sérieux qui convient, nous ne pouvons ignorer que, parfois, les arbres cachent la forêt et que nombre d'entre nous trébuchent sur des racines. On ne sait pas toujours très bien par où et comment commencer, et, par conséquent, un premier contact malheureux avec l'ordinateur peut se révéler définitivement négatif. Vous l'avouerez, c'est d'autant plus regrettable que cela pourrait être évité.
C'est tout le "pourquoi" de ce livre et du second ouvrage qui le suivra d'ici peu.
Ce premier tome traite donc du Junior Computer, un micro-ordinateur d'une éclatante jeunesse. Il est équipé du microprocesseur 6502 dont la réputation n'est plus à faire. Notre choix délibéré d'un processeur élaboré découle de notre conviction que, plus vite on apprend à jouer du piano sur un véritable instrument et non sur un jouet d'enfant, plus vite on acquiert la maîtrise du jeu. D'autre part, le système de base, tel qu'il est décrit dans ce livre, réserve de nombreuses possibilités d'extension, qui, au moment opportun, lorsque le constructeur en éprouve la nécessité, peuvent être exploitées partiellement ou en totalité.
Le Junior Computer est un micro-ordinateur à construire soi-même, ce qui, évidemment, a une heureuse incidence sur le prix de revient. Les objections que certains auraient pu soulever sur ce dernier point, tombent d'elles-mêmes, nous semble-t-il. La conception du Junior Computer met l'accent sur l'homogénéité du système; en outre, point n'est besoin de faire preuve d'une adresse acrobatique dans le maniement du fer à souder. L'un de nos objectifs essentiels a été de mettre à la disposition du constructeur la majeure partie du chapitre 1 du tome 1. Ce chapitre est également une introduction générale aux micro-ordinateurs ainsi qu'aux ordinateurs plus "sophistiqués".
Le chapitre 2 est consacré aux nombres et aux calculs que nous connaissons très bien depuis l'école primaire. Dès que nous avons assimilé les règles et les processus grâce auxquels le Junior Computer manipule les chiffres et effectue les opérations arithmétiques et logiques, nous sommes invités à aborder le chapitre 3 où nous découvrons les secrets de la programmation. Finalement, le chapitre 4 nous propose un certain nombre de programmes-types. En outre, les chapitres 3 et 4 nous offrent l'occasion de faire fonctionner utilement le Junior Computer tout nouvellement construit. Rien de tel que la mise en pratique immédiate de l'enseignement théorique, grâce aux programmes qui nous permettent de récolter les premiers fruits de nos efforts.

S'il nous fallait définir d'une certaine manière la philosophie de cet ouvrage, nous dirions que c'est une erreur de confondre "simple" et "superficiel". Et encore, qu'il n'est pas judicieux de vouloir tout à la fois. D'où le dosage de la communication des connaissances à acquérir et la répartition de celles-ci en deux tomes. Dans ce premier livre, nous vous indiquons tout ce qu'il faut savoir pour programmer "sans confusion". Ainsi serez-vous conduit tout naturellement jusqu'au livre 2, dans lequel vous apprécierez la gamme des possibilités nouvelles offertes par le Junior Computer, auquel vous serez déjà parfaitement accoutumé.
Cessons donc de "philosopher" et permettez-nous de vous exprimer notre espoir d'avoir contribué, avec ce premier ouvrage, à susciter et à satisfaire votre désir de posséder la maîtrise du micro-ordinateur et de la micro-informatique.

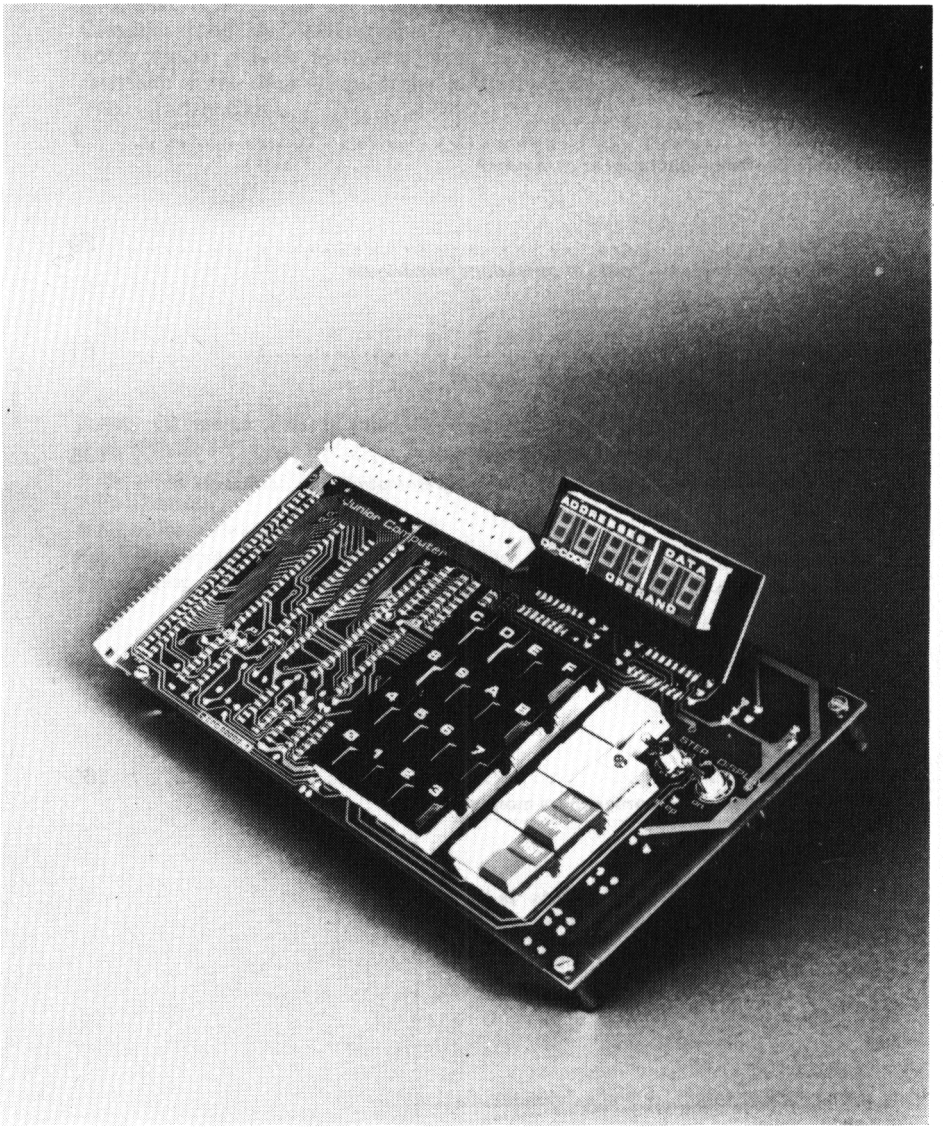
Les auteurs

Junior Computer 2 est la suite de ce livre. Il est consacré:

- au boîtier I/O type 6532 du Junior Computer et à sa programmation,
- au programme moniteur et à toutes les possibilités qu'il offre à l'utilisateur,
- à l'éditeur hexadécimal,
- à l'assembleur hexadécimal,
- au listing du programme moniteur.

Les circuits imprimés présentés dans ce livre sont distribués par Publitronic. Pour de plus amples informations, le lecteur est prié de se reporter au dernier numéro de la revue "ELEKTOR".

Table des matières


Junior Computer
ADDRESS
DATA
OPCODE
OPERAND
C D E F
8 9 A B
4 5 6 7
0 1 2 3
STEP
GO

1

Premier contact

Pourquoi et comment?

S'agissant du Junior Computer, le terme "Junior" a eu à nos yeux au moins autant d'importance que le terme "Computer". Cela signifie que, dès l'introduction de ce chapitre et jusqu'à l'achèvement de cet ouvrage, notre intention est bien d'écarter tous les malentendus et les préjugés, sans en créer de nouveaux. Par conséquent, ne vous laissez pas impressionner par notre (micro)ordinateur! Que les composants, présentés dans ce premier chapitre, et le logiciel, qui fera l'objet des chapitres suivants, ne vous découragent pas!

En abordant la lecture de ce premier livre, que nos lecteurs soient intimement persuadés, qu'en réalité, un micro-ordinateur, et spécialement le Junior Computer, est un appareil simple.
Il se peut que certains d'entre eux soient décontenancés par une apparente complexité, qui ne résulte sans doute que de la part trop large faite, à priori, à une multitude de composants électroniques. Qu'ils se rassurent, les circuits intégrés actuels rassemblent sur une seule petite "puce" un tel nombre de transistors, de résistances, de condensateurs, que la réalisation pratique s'en trouve d'autant simplifiée. Voilà une excellente raison de ne pas se priver du plaisir de construire soi-même son ordinateur, après avoir facilement assimilé la structure et le fonctionnement du circuit.
Le but d'un micro-ordinateur est l'exécution des tâches logiques qui lui sont assignées par l'utilisateur, cela par l'intermédiaire d'un programme. Et, par conséquent, le défi auquel répond le constructeur amateur ne réside pas tant dans la pénétration du système électronique que dans l'invention d'instructions habilement élaborées et judicieusement agencées, à la hauteur des capacités de l'ordinateur.
Il importe donc de considérer le micro-ordinateur, et, dans le cas qui nous occupe, le Junior Computer, comme une sorte de "boîte noire" (ou de toute autre couleur) nécessitant que notre intérêt se concentre particuliè-

rement sur ses éléments tournés vers le monde extérieur: le clavier et l'affichage.
Car la question essentielle est celle-ci: comment puis-je, par l'intermédiaire du clavier, mettre en œuvre l'ordinateur, et quel sera le résultat de mon travail, inscrit lumineusement par ses afficheurs?

Pas de soucis injustifiés

Avant que vous ayez serré les dernières vis du coffret et qu'échappe ainsi à vos regards le contenu du Junior Computer, il vous faudra d'abord le construire. L'essentiel de ce premier chapitre sera consacré à la réalisation et au fonctionnement de l'ordinateur. Après quoi, les chapitres qui suivront traiteront de manière approfondie de la programmation et des extensions possibles. Mais, bien entendu, il faut préalablement disposer de cette "boîte noire", qui, après avoir été raccordée au secteur, sera entièrement livrée à vos initiatives.
Certains se demanderont si nous ne présumons pas trop de leurs capacités en leur proposant de construire eux-mêmes leur Junior Computer. Evidemment, non. Et nous serons là pour vous aider d'un bout à l'autre, par une description détaillée de la réalisation. Au moment de la mise au point finale, nos conseils, nos "tours de main" contribueront à vous faire franchir aisément cette dernière étape. Ensemble, cela ira bien mieux, n'est-il pas vrai...

Un peu de technique

Le jeu de construction est l'un des premiers que l'être humain réussisse à maîtriser, et c'est aussi par le biais d'un assemblage de quelques blocs que vous allez réaliser votre Junior Computer. La figure 1 est l'illustration d'une telle juxtaposition de trois blocs et de trois types de liaisons les interconnectant. Un ordinateur fonctionne à l'aide d'informations qui lui sont fournies sous une forme qu'il peut assimiler. Il s'agit de *données*

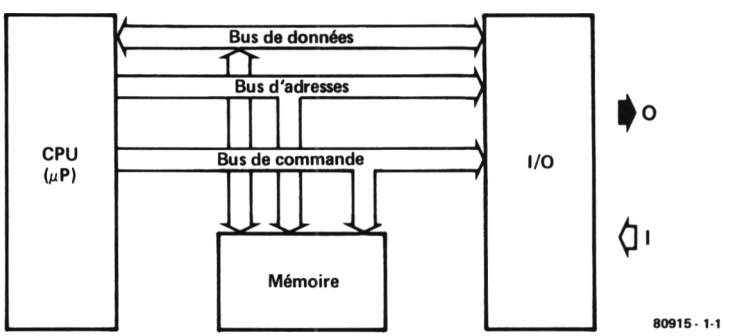


Figure 1. Le schéma fondamental d'un ordinateur regroupe trois blocs et les trois bus les interconnectant.

exprimées par des impulsions électriques appliquées selon un code numérique. L'échange d'informations entre les divers blocs ainsi que leur transfert vers et à partir de ceux-ci, se fait par l'intermédiaire du *bus de données*.
S'il ne communique pas avec le monde extérieur, un ordinateur est sans objet; en ce qui le concerne, ce monde peut être représenté par le clavier d'un télé-imprimeur, l'écran d'une console de visualisation, un affichage, mais aussi, une raffinerie. A cet égard, le point de jonction (interface) entre le Junior Computer et l'extérieur est constitué par le bloc I/O (de Input/Output = entrée/sortie), grâce auquel est régulée la circulation des données en provenance de et allant vers les appareils ou installations situées à sa périphérie (et que l'on désigne souvent sous le nom de "périphériques"). Les données présentes sur le bus de données sont ou vont être traitées par le bloc CPU (Central Processing Unit = Unité Centrale de Traitement), que l'on considère aussi volontiers comme le cerveau central de l'ordinateur, encore que son intelligence soit toute relative. Toutes les instructions sont gérées par le CPU. Dès qu'une instruction déterminée a été exécutée, il lui faut s'inquiéter du contenu de l'instruction suivante et pour cela il va explorer une section de la *mémoire* où les instructions sont consignées en code. Cet ensemble ou jeu d'instructions correspondant à une application précise de l'ordinateur s'appelle le *programme*. Lorsque le CPU est associé avec une section de mémoire ou avec une partie du bloc d'entrée/sortie sur une "puce" ("chip" en anglais), on parle alors d'un *circuit intégré* constituant un *microprocesseur*.
C'est donc dans la mémoire que l'ordinateur conservera le souvenir de ce que vous lui aurez indiqué de faire. Il n'est pas doté d'une intelligence capable d'initiatives basées sur un mode de réflexion du genre: "Et maintenant, que vais-je bien pouvoir imaginer?". Son type de raisonnement s'apparente plutôt à: "J'ai tel problème à résoudre, quel processus vais-je adopter?". La mémoire ne sert pas seulement au stockage du programme; il s'y trouve également consignées des données nécessaires à l'exécution des instructions et d'autres, qui, à la suite de cette opération, y sont emmagasinées afin de pouvoir servir ultérieurement.
Dans l'ordinateur, les données sont la matière première des manipulations, ainsi d'ailleurs que les inscriptions, consignées en code, relatives à ces manipulations (c'est le programme). Cette matière première est élaborée sous forme d'impulsions codées et il existe un transfert de données entre les trois blocs de base. Le CPU, quant à lui, peut aller chercher les données qui lui sont indispensables pour l'exécution d'une instruction; l'information transite également vers et depuis la périphérie du système. La sélection des données, ainsi que celle de certaines entrées et sorties, s'opère sous la forme d'un adressage: en tel endroit, à cet instant précis, est présente telle donnée nécessaire à l'exécution de telle phase du programme. Ou encore: telle donnée doit être introduite ou acheminée à l'extérieur, par l'intermédiaire de tel point d'entrée/sortie (port). Cette circulation des données se fait grâce au *bus de données*.
Pour chaque instruction à exécuter, l'indication codée de l'emplacement de la mémoire qui constitue le point de départ ou d'arrivée du transfert de donnée correspondant à l'instruction, transite par le *bus d'adresses*.
Finalement, il existe un troisième bus, le *bus de commande* (en anglais,

"control bus"). Son rôle est de réguler le déroulement d'un certain nombre de processus internes. C'est par lui que cheminent des signaux de commande auxiliaires réglant la nature et la direction du transfert des données. Ils proviennent du CPU (microprocesseur ou μP), soit à son "initiative", soit sous l'effet d'une influence externe. La régulation des sous-programmes successifs, et, par conséquent, celle des instructions, fait aussi partie des tâches internes qu'il assume. A cette fin, il est périodiquement excité par un dispositif dont l'action peut se comparer à celle du cervelet entretenant les pulsations cardiaques.

Et maintenant, rien que de la technique

Du schéma très général de la figure 1, nous passons à celui du Junior-Computer, en figure 2. Commençons par examiner les bus. Le bus d'adresses comporte 16 fils de connexion. Chaque fil peut, indépendamment des autres, prendre deux états (électriques). Ce qui veut dire, qu'il faudra au moins qu'existent deux niveaux de tension différents. Cet ensemble de 16 fils de connexion va permettre d'obtenir jusqu'à 2^{16} (soit, 65536) états différents qui correspondront à plus de 65000 emplacements de la mémoire constituant les points d'arrivée ou de départ des transferts de données. Chaque fil achemine 1 bit (autrement dit, 1 donnée), d'où la présence sur la figure 2 de 16 bits pour le bus d'adresses. Le nombre de bits est en rapport avec la quantité d'informations codées susceptibles d'être transférées. Nous reviendrons sur ce sujet dans le second chapitre.
Par le bus de données, qui se compose de 8 fils conducteurs, se fait le transfert des données. Celles-ci circulent dans les deux sens; c'est la raison

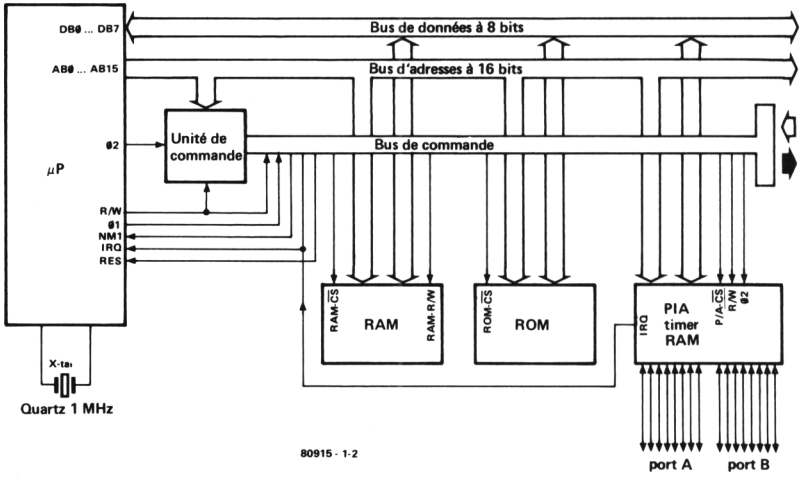


Figure 2. Le schéma de la figure 1 a évolué pour devenir celui d'un véritable ordinateur, le Junior Computer.

pour laquelle on dit que le bus de données est bidirectionnel. Mais, le transit ne s'effectue jamais dans les deux sens en même temps; c'est la circulation à sens unique alterné. Le signal de lecture/écriture (R/W = Read/Write) du bus de commande influence la direction du transfert. Il détermine le comportement d'étages tampons (buffers) incorporés dans le bus de données (il en existe un pour chaque conducteur) à l'égard de l'état des extrémités du conducteur, et plus précisément, il permet que telle broche devienne "entrée" tandis que la broche connectée à l'autre bout du conducteur devient automatiquement "sortie". En outre, chaque connexion peut être mise "hors service". D'un point de vue technique, c'est ce qu'on appelle la "logique à trois états" (tri-state logic).

Les mémoires

Il existe deux catégories principales de mémoires qui toutes deux sont indispensables au bon fonctionnement d'un ordinateur. C'est là l'origine des deux blocs RAM et ROM de la figure 2, à la différence du bloc unique de la figure 1. Il y a des données dont la présence est permanente, par exemple, celles faisant partie du *programme moniteur*, qui est le programme de base de l'ordinateur. D'autre part, certaines données sont variables, par exemple celles d'un programme que l'utilisateur compose, et d'autres encore doivent être modifiées avant qu'elles soient utilisables. En outre, il existe des variables d'entrée ou de sortie du programme.
La donnée permanente est emmagasinée dans la mémoire à seule fin de pouvoir y être recherchée indéfiniment. La donnée non permanente peut être aussi bien inscrite que lue. Car, lorsque nous parlons de mettre en mémoire ou de rechercher en mémoire une donnée, nous entendons par là qu'on l'y "inscrit" ou qu'on la "lit". En fait, lorsqu'une donnée est écrite en mémoire, c'est généralement une "copie" d'elle-même qui va être utilisée. Certaines mémoires ne peuvent être que lues, d'où le symbole ROM, de Read Only Memory [1] (mémoire à lecture seule). La mémoire du programme de base du Junior Computer est assimilable à une ROM. Par contre, d'autres mémoires, les RAM, de Random Access Memory (mémoire à accès aléatoire) peuvent être inscrites, lues ou effacées; c'est le cas de la mémoire de travail. Voici l'une des raisons du transfert de données bidirectionnel sur le bus de données: le même signal de

---

[1] *En dehors de la ROM, dont le programme a été enregistré définitivement par le fabricant au moment de la fabrication et dont le contenu ne peut plus être modifié par l'utilisateur, il existe des versions dérivées. L'EPROM, de Electrically Programmable ROM (ROM programmable électriquement), peut être programmée par l'utilisateur à l'aide d'un programmateur d'EPROM, après effacement du contenu précédent sous l'action des ultra-violets. La ROM n'est pas reprogrammable, à la différence de l'EPROM et de l'EAROM. De même, la PROM, de Programmable ROM, ne l'est pas. Elle est livrée vierge de toute inscription par le fabricant et c'est l'utilisateur qui charge les données par la destruction de "jonctions fusibles" lesquelles sont des connexions destructibles. Le programme chargé ne peut plus être modifié par la suite.*

lecture/écriture qui fixe le sens du transfert, détermine également le fait que la RAM sera lue (sortie de données) ou qu'on y écrira (entrée de données).
En fait, comme nous le disions, la lecture concorde avec le prélèvement d'une donnée, mais celle-ci reste à l'emplacement qu'elle occupe dans la mémoire. Elle n'est que copiée, et c'est cette copie qui est déposée sur le bus de données pour y transiter. Lorsqu'on lit un livre, il y a transfert de l'information, codée sous forme de mots et de phrases à l'aide des caractères; la lecture du livre achevée, les lettres n'ont pas disparu du papier. En ce point, il y a lieu de faire une réserve concernant les RAM, par exemple, car, à supposer que l'on débranche l'ordinateur ou que l'alimentation du réseau soit coupée par une panne, leur contenu est perdu. C'est la raison pour laquelle, lorsqu'on veut s'assurer de la permanence des données stockées dans une RAM, le meilleur moyen est de les copier dans une *mémoire auxiliaire* telle qu'un floppy disc (disque souple) ou une bande magnétique (cassette).

Le bloc d'entrée/sortie I/O

Le bloc d'entrée/sortie maintient le contact avec le monde extérieur (et, plus l'ordinateur est petit, plus le monde extérieur est important). Dans le schéma synoptique de la figure 2, nous retrouvons ce bloc sous la forme du PIA, de Peripheral Interface Adapter (interface d'adaptation pour périphériques). Certes, les trois bus se dirigent également vers l'extérieur (voir les trois flèches à la droite de la figure 2) mais c'est en vue de l'extension ultérieure du Junior Computer lui-même, donc de sa "capacité interne". En tout cas, pour ce qui nous concerne, le contact avec l'extérieur est assuré physiquement par le clavier et l'affichage.
Tout comme avec les mémoires, le transfert des données doit pouvoir s'effectuer dans les deux sens. Le bus de données, avec ses huit conducteurs (pour 8 bits), se prolonge vers l'extérieur sous la forme de deux faisceaux de chacun huit fils constituant deux ensembles que l'on appelle les *ports* A et B. La sélection des ports s'opère par l'entremise du bus d'adresses. Chacune des connexions des ports peut être sélectionnée à un moment donné, indépendamment des 15 autres, soit en tant qu'entrée, soit en tant que sortie.
Dans le PIA sont stockées (temporairement) des données à destination ou en provenance du monde extérieur avec lequel le Junior Computer veut communiquer. A cet effet, il y a un certain nombre de bistables réunis en un registre et fonctionnant dans les deux directions, toujours sur le principe du sens unique alterné. Le bus de commande délivre les signaux qui provoqueront la lecture ou l'écriture des données dans le registre.

Le microprocesseur

Animé par un "pace maker", le générateur de signaux d'horloge (auquel est associé un quartz externe: 1 MHz XTAL), le microprocesseur commande tout le système par l'intermédiaire des trois bus. Le générateur d'horloge délivre deux signaux, Ø1 et Ø2, pour chaque instruction à exécuter, qui jouent un rôle essentiel dans la préparation des bus d'adresses

et de données respectivement. L'horloge, quant à elle, est du type biphasé. C'est en tout cas à ce niveau que s'élabore le nécessaire "métabolisme" des données.
Le *compteur ordinal*, PC (Program Counter), est un autre composant du microprocesseur. Il contient l'adresse de la prochaine instruction à exécuter, qu'il déposera sur le bus d'adresses. Les adresses proviennent d'autres composants du microprocesseur. L'état d'un autre compteur interne spécial, qui gouverne l'exécution de chaque instruction suivante, est déterminant pour que soit extraite l'adresse de la prochaine instruction à exécuter. Les adresses, que ce soit en tant que matières premières, semi-produits ou produits finis du programme, apparaissent directement ou comme résultat des manipulations effectuées par le programme.

Les interruptions

Il nous faut encore examiner trois signaux mentionnés sur la figure 2, RES, IRQ et NMI. Du signal RES, de Reset (Remise à zéro), nous dirons brièvement qu'il permet de lancer le Junior Computer dans l'exécution d'un programme ou d'un sous-programme, après mise sous tension du système. Les signaux IRQ, de Interrupt ReQuest (demande d'interruption) et NMI, de Non Maskable Interrupt (interruption non masquable) permettent d'interrompre de manière constructive l'exécution d'un programme en cours, afin d'introduire des instructions plus ou moins urgentes.
L'initiative d'une telle demande vient de l'extérieur et émane d'un organe d'entrée/sortie. L'attention de l'ordinateur est mobilisée par une pression exercée sur une touche du clavier, ce qui signifie que l'utilisateur a simplement quelque chose à lui dire, ou qu'un appareil périphérique déterminé, par exemple, une imprimante, demande une intervention rapide. Dès qu'une telle "impulsion d'interruption" est perçue, l'ordinateur interrompt le programme où il en était, après avoir achevé l'exécution de l'instruction en cours. En même temps, le microprocesseur veille à la sauvegarde du contenu du compteur ordinal et de divers registres internes. Ceci fait, la routine d'interruption peut être exécutée. Dès qu'elle est achevée, le programme interrompu reprend. Il est fréquent qu'il y ait simultanément plusieurs demandes d'interruption. Dans ce cas, chacune d'elles se voit assigner un niveau de priorité déterminé. Plus le degré de priorité est élevé, et plus rapidement la demande d'interruption sera satisfaite. Le fait que l'instruction IRQ puisse être ignorée par le programme lui-même ou par son intermédiaire (grâce à l'étiquette "ne pas interrompre"), constitue une différence considérable entre les deux instructions IRQ et NMI qui, elle, ne peut l'être (elle est non masquable).

En outre...

Dans le bloc PIA de la figure 2, est présent également un timer (temporisateur, ou encore générateur d'intervalles de temps programmables) sur lequel nous reviendrons au chapitre 5.
Le bloc "Unité de commande" (Control logic) comporte un décodeur d'adresses. C'est grâce à lui que seront engendrés les différents signaux Chip Select (sélection du circuit) qui préparent à l'emploi une mémoire déterminée ou un registre du PIA, en vue des opérations lecture/écriture.

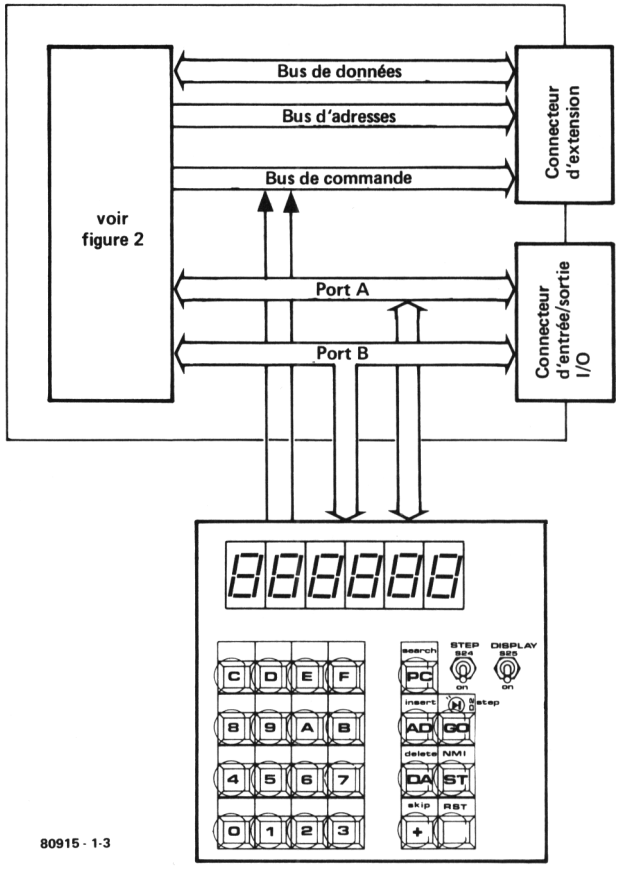


Figure 3. Nouvelle évolution par rapport au schéma de la figure 2: le clavier et l'affichage rendent possibles les contacts avec le monde extérieur, grâce aux entrées (Input = I) à l'aide des touches et aux sorties (Outpout = O) sur les afficheurs, de données (data) écrites en notation hexadécimale.

Les yeux et les mains

Les points de contact entre l'ordinateur et son utilisateur complètent le Junior Computer. C'est ce qu'illustre la figure 3; le schéma synoptique de la figure 2 se voit complété par le clavier et l'affichage, grâce à quoi l'utilisateur se servira de ses yeux et de ses mains pour converser avec l'ordinateur.

En outre, on constate que les trois bus sont accessibles via le "connecteur d'extension"; les ports A et B le sont par l'intermédiaire du "connecteur d'entrée/sortie" I/O. Le clavier hexadécimal, en tant qu'organe d'entrée (Input = I) standard et l'affichage hexadécimal, en tant qu'organe de sortie

(Output = O) standard, constituent les intermédiaires les plus simples (et les moins coûteux) permettant les échanges entre l'utilisateur et l'ordinateur. Il existe bien d'autres dispositifs d'entrées/sorties grâce auxquels la qualité, la densité du contenu de la conversation sont sensiblement accrues. C'est là un aspect très intéressant, mais qui exige sa contrepartie en argent et en savoir-faire. Cependant, il ne nous a pas échappé, car nous estimons que le Junior Computer doit pouvoir évoluer à l'unisson de l'évolution des capacités de ses utilisateurs.
C'est pourquoi le connecteur d'extension pourra être utilisé à l'accroissement de la capacité de la mémoire, de telle sorte que, par exemple, l'on puisse composer des programmes plus importants et plus sophistiqués. D'autre part, ce connecteur à 64 broches est entièrement compatible avec le bus du SC/MP d'Elektor. Quant aux seize fils conducteurs des ports A et B, ils sont raccordés à un connecteur à 31 broches.
Le clavier et l'affichage sont connectés à l'ordinateur, par l'intermédiaire des ports A et B; au niveau du port A, le transit des informations est bidirectionnel, alors qu'il est unidirectionnel par le port B. Deux signaux sont acheminés par le bus de commande; il s'agit de RES et NMI, correspondant respectivement aux touches RST et ST situées à la partie supérieure gauche du clavier.
A l'usage, nous allons apprendre à tirer le meilleur parti des possibilités offertes par le clavier et l'affichage du Junior Computer. Certes, il y faudra un peu de temps, mais cela viendra vite. Préalablement, nous nous consacrons maintenant à l'examen de la totalité de l'équipement électronique de notre Junior Computer; c'est ce qu'on appelle le "hardware". Après que nous en ayons achevé l'assemblage, il nous restera à nous lancer dans l'apprentissage de la programmation, dans la réalisation de programmes, ce que les spécialistes désignent sous le nom de "software" ou logiciel.

Entrons dans le détail

La figure 4 donne une vue détaillée de l'ensemble de l'équipement électronique du Junior Computer et nous commençons l'étude du schéma de principe par le microprocesseur IC1. C'est un circuit intégré dont la réputation n'est plus à faire. Ce 6502 est particulièrement apprécié pour la richesse de son jeu d'instructions (Instruction set), ainsi que pour les vastes possibilités offertes par le software qui lui est associé.

L'horloge et son générateur

Le microprocesseur a besoin d'une base de temps précise et celle-ci lui est délivrée par un générateur de signaux d'horloge regroupant N1, R1, D1, C1 et un quartz dont la fréquence est de 1 MHz. Chaque microseconde, cet ensemble veille à ce que soit fourni un train d'impulsions électriques formé de deux signaux, Ø1 et Ø2, nécessaires à la détermination de l'état des entrées et sorties des bus d'adresses et de données. Grâce au quartz, la fréquence de ce train d'impulsions est précisément d'un million de fois par seconde. Certes, le quartz n'est pas absolument indispensable, mais alors, une grande précision ne pourrait être atteinte.

Le transit des "1" et des "Ø"

Le bus d'adresses a pour point de départ le microprocesseur IC1 et se compose des lignes AØ . . . A15. A l'égard du bus de données, lignes DØ . . . D7, IC1 est à la fois point de départ et point d'arrivée. L'état électrique du bus d'adresses et celui du bus de données recèle le code de l'adresse, d'une part, de la donnée, d'autre part. Comment cela se peut-il? Constatons d'abord que l'état électrique d'un fil conducteur peut se caractériser éventuellement, soit par la présence d'une tension à ses extrémités, soit, au contraire, par l'absence de toute tension. Décidons que la présence d'une tension sera représentée par un "1" et que l'absence de tension se traduira par un "Ø". Nous avons là les deux caractères d'un code à deux chiffres. Ces deux chiffres disposés selon un certain ordre de succession pourront représenter un mot dont la longueur correspondra au maximum au nombre de fils conducteurs porteurs ou non d'un niveau de tension. Prenons, par exemple, les 16 conducteurs du bus d'adresses dans l'ordre de succession A15 . . . AØ et les huit conducteurs du bus de données dans l'ordre D7 . . . DØ. Supposons, qu'en un instant donné, l'état des tensions et des absences de tension soit tel que nous ayons respectivement les ordres de succession 1ØØ1Ø111Ø11Ø1ØØ1 pour le bus d'adresses et 1Ø1ØØ1Ø1 pour le bus de données. Nous obtenons effectivement deux mots codés à l'aide de "1" et de "Ø", qui nous livrent, d'une part l'adresse, d'autre part la définition d'une information. L'alphabet de notre code ne comporte que deux lettres, "1" et "Ø", et c'est pourquoi nous disons que c'est un code binaire. Chacune de ces lettres est appelée un "bit", de l'expression "binary digit" (chiffre binaire).

Organisation de la mémoire

Pour des raisons de commodité dans la manipulation des informations, il est souvent préférable de considérer les bits par groupements de 8, chaque groupe de 8 bits formant un "octet" (byte, en américain). Cette notion va trouver immédiatement son application dans notre exposé relatif aux sections de la mémoire que l'on discerne dans le schéma synoptique de la figure 4. L'EPROM, IC4, constitue la mémoire permanente de programme, alors que les RAM, IC4 et IC5, forment la mémoire de travail dont le contenu pourra varier. Chaque fois, le transfert des données s'effectue par groupe de 8 bits (soit, 1 octet), ce qui fait que chaque emplacement de mémoire peut accueillir 1 octet, l'équivalent de l'état électrique des huit fils conducteurs du bus de données consigné (écriture) ou copié (lecture) tel qu'il était en un instant donné. L'EPROM permet l'inscription de 1024 octets. Dans le langage des informations, on dit qu'elle a une capacité de 1K-octet, ou encore, 1 kilo-octet. Cela correspond précisément à 10 lignes du bus d'adresses, AØ . . . A9, donnant 2^{10} = 1024 adresses différentes.

Chacune des RAM peut contenir 1024 demi-octets, de chacun 4 bits. A elles deux, elles ont donc une capacité totale de 1024 octets. Pour une adresse déterminée. les quatre premiers bits situés les plus à gauche, (D7 . . . D4), sont inscrits dans IC5; les quatre autres bits du mot, situés les plus à droite, (D3 . . . DØ), le sont dans IC4.

Les deux types de mémoires reçoivent encore du bus de commande

des signaux de sélection, par exemple. L'EPROM et les deux RAM sont connectées aux mêmes lignes du bus de commande, AØ . . . A9. Se pose alors la question de savoir comment s'opère la distinction entre les deux mémoires? Elle est réalisée grâce aux signaux $\overline{CS}$ (Chip Select), qui, provenant du décodeur d'adresses, IC6, sont acheminés, via le bus de commande, par la ligne K7 pour l'EPROM et par la ligne KØ pour les RAM. En supposant qu'un "1" (tension) soit présent sur une ligne $\overline{CS}$, les entrées de données de la mémoire à laquelle elle est connectée sont déconnectées du bus de données. Si une ligne $\overline{CS}$ est à l'état "Ø" (absence de tension, ou 0 Volt), alors la mémoire sur laquelle elle est branchée participe au transfert des données.
Dix lignes d'adressage étant communes aux deux types de mémoire, il reste donc six lignes grâce auxquelles $2^6 = 64$ blocs de chacun 1K-octet peuvent être adressés, ce qui donne une capacité totale de 64 K.
Par conséquent, si l'on disposait de 64 lignes $\overline{CS}$, il serait possible de sélectionner la totalité des blocs mémoire dont l'adressage se ferait par les 10 lignes AØ . . . A9. Cependant, le Junior Computer, en version standard, n'en compte que 8. Elles acheminent les signaux KØ . . . K7 provenant du décodeur d'adresses IC6, auquel sont connectées les trois lignes d'adresses A10 . . . A12. Trois des huit signaux $\overline{CS}$ sont exploités intérieurement. Outre les deux déjà cités, K7 et KØ, le troisième est K6 qui sélectionne la RAM, IC3, intégrée dans le PIA. Nous donnons ci-dessous une sorte de résumé sous forme de tableau:

A15 . . . A13	A12	A11	A10	A9 . . . AØ	actif	bloc mémoire:
X	Ø	Ø	Ø	X	KØ	1K RAM (IC4,IC5)
X	Ø	Ø	1	X	K1	1K externe } RAM,
X	Ø	1	Ø	X	K2	1K externe } ROM
X	Ø	1	1	X	K3	1K externe } ou
X	1	Ø	Ø	X	K4	1K externe } RAM
X	1	Ø	1	X	K5	1K externe } du PIA
X	1	1	Ø	X	K6	RAM du PIA (IC3)
X	1	1	1	X	K7	1K EPROM (IC2)

Le symbole X signifie "quelconque", ce qui veut dire que les "syllabes" à trois lettres de la première colonne et les syllabes à 10 lettres (bits) de la cinquième colonne peuvent se composer de combinaisons quelconques de "1" et de "Ø".
Par conséquent, sur une capacité totale de 64K, 8K seront traités par le décodeur de la version standard du Junior Computer. Pour obtenir une capacité plus importante de décodage, une extension du décodeur d'adresse est obligatoire, et sera extérieure au système standard. Mais il est important de bien considérer que, même dans sa version standard, le Junior Computer a une capacité d'adressage plus que suffisante pour une très sérieure "mise en train".
La mise en exploitation de la RAM nécessite qu'on lui délivre un signal de commande qui fasse la différence entre la lecture et l'écriture. Ce signal, R/W (Read/Write), sera un "1" pour la lecture, et un "Ø" pour l'écriture; il apparaîtra à la sortie de la porte N6 et sera le résultat d'une combinaison d'un signal R/W émis par le microprocesseur et d'une impul-

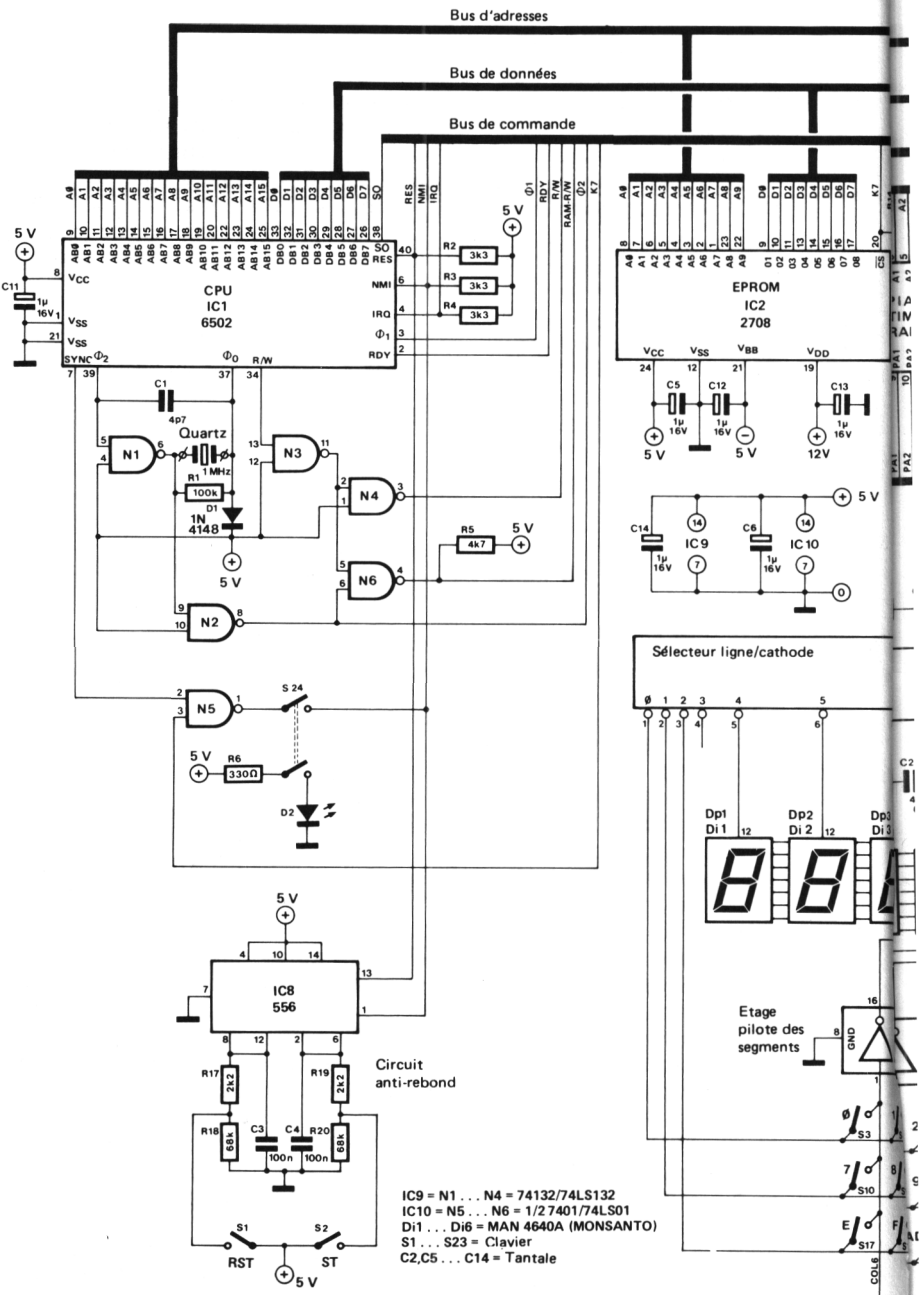
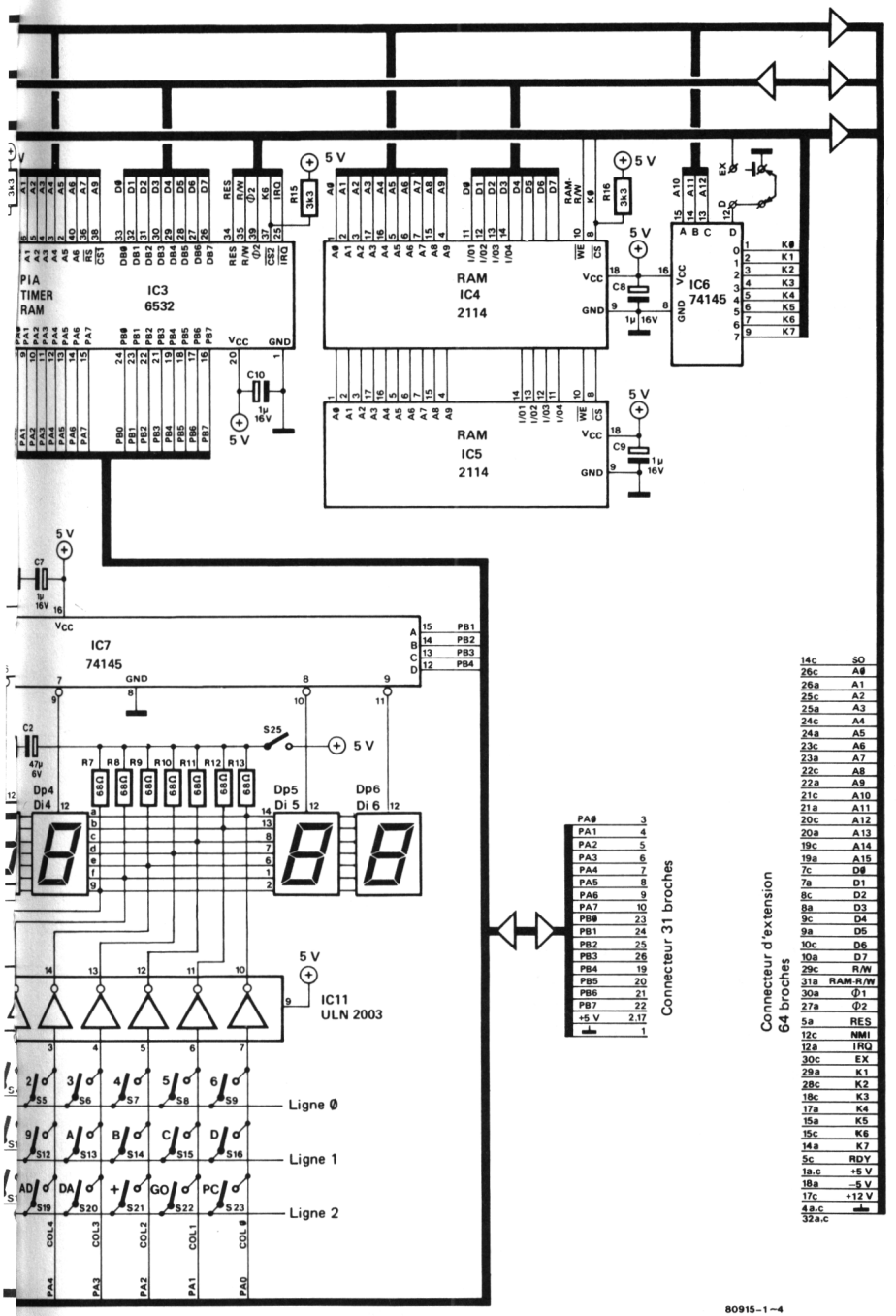


Figure 4. Schéma du circuit électronique du Junior Computer. Pour des raisons de lisibilité, chacun des trois bus est représenté sous la forme d'une ligne de connexion unique.


PIA
TIMER
RAM
IC3
6532
RAM
IC4
2114
RAM
IC5
2114
IC6
74145
IC7
74145
IC11
ULN 2003
5 V
Vcc
GND
Dp4
Di4
Dp5
Di 5
Dp6
Di 6
Ligne Ø
Ligne 1
Ligne 2
Connecteur 31 broches
Connecteur d'extension
64 broches
80915-1-4

sion d'horloge Ø2, ce qui garantit qu'aucune tentative d'écriture de donnée ne puisse être faite tant que le bus de données n'est pas prêt à l'acheminer.

Fonctions du bus de commande

Nous avons déjà mentionné un certain nombre de signaux de commande. Le moment est venu d'examiner les autres. Avec le signal de reset, (remise à zéro) RES, le microprocesseur IC1 et le PIA IC3 sont forcés dans une position de départ déterminée. Cela résulte du passage de l'état "1" à l'état "Ø" des broches RES des deux circuits intégrés. Normalement, ces deux points sont portés à l'état "1" par l'intermédiaire de la résistance de polarisation (pull-up) R2. L'instruction de reset est transmise par une pression exercée sur la touche RST (représentée par S1, dans le schéma de la figure 4). Un timer intégré à IC8 est intercalé pour assurer la neutralisation des rebonds de touche.
L'entrée NMI de IC1, portée normalement à l'état "1" par l'intermédiaire de R3, est mise à "Ø", soit sous l'influence d'une instruction externe transmise via le connecteur d'extension, soit sous celle de deux interventions internes. Ce peut être le résultat d'une pression sur la touche STOP (représentée par S2, dans la figure 4), qui, grâce à l'action de l'autre moitié de IC8, délivre un signal de bien meilleure qualité parce que moins affecté par les rebonds. Ou bien, une pression sur la touche STEP (S24, dans le schéma de la figure 4) fait passer la sortie de la porte N5 de l'état haut ("1") à l'état bas ("Ø"), ce qui est indispensable pour l'exécution du programme "pas à pas". Cependant, si les adresses sont destinées à l'EPROM, où est inscrit le programme moniteur, les possibilités d'interruption doivent être bloquées, d'où la présence de K7 à l'entrée de N5. Si celle-ci est à "Ø", jamais sa sortie ne pourra passer de "1" à "Ø".
A supposer que les broches IRQ de IC1 et IC3 soient à "Ø", le programme se trouve également interrompu, tant que lui-même n'en décidera pas autrement, car, nous l'avons vu au paragraphe "Interruptions", il a la faculté de le faire. Sans intervention externe, l'exécution d'une requête d'interruption n'est possible que grâce à la programmation du timer associé à la RAM du PIA. Il s'agit alors d'une interruption software.
D'autre part, le bus de commande achemine également les signaux d'horloge Ø1 et Ø2, ce dernier intervenant dans le fonctionnement du PIA. Il existe aussi une ligne R/W, non influencée par le signal Ø2, connectée à la broche R/W de la RAM. Le signal transitant par cette ligne détermine le sens du transfert des données et influence l'état des buffers bidirectionnels situés à l'intérieur de IC1 et IC3, en aval des connexions du bus de données. Enfin, existent également les lignes RDY et SO, non utilisées intérieurement, mais qui jouent un rôle dans le cas d'une éventuelle connexion de cartes RAM dynamiques externes. Quant à la ligne EX (voir IC6), elle est indispensable à la réalisation d'une extension externe de la capacité du décodeur d'adresses.

L'interface I/O

Partant du microprocesseur IC1, le bus de données bidirectionnel s'étend jusqu'aux portes A et B, via l'organe de connexion multiport qu'est le

PIA. A chaque port correspondent un registre d'entrée/sortie et un registre de direction des données de 8 bits, dans lequel est spécifiée la configuration d'une ligne en entrée, "Ø", ou en sortie, "1". Le contenu du registre de direction est déterminé par le software, en conséquence, par le programmeur.
Le signal R/W fixe la direction du transfert de données. Si c'est un "1", la donnée est lue dans le PIA; un "Ø" provoque l'écriture de celle-ci.
Une RAM est également intégrée dans le PIA, mais sa capacité n'est que de 128 octets. Ajoutés aux 1024 octets des RAM IC4 et IC5, cela représente un total de 1152 octets. Les sept lignes du bus d'adresses AØ . . . A6 suffisent à l'adressage des 128 emplacements mémoire. Le signal RS fait fonction d'aiguillage électronique; grâce à lui, on modifie l'orientation du transfert des données depuis ou vers la RAM, et depuis ou vers l'un des ports. Lorsque le bit est à "0", la RAM est connectée; si le bit est à "1", ce sont les ports. La ligne d'adresse A7 joue le rôle d'aiguilleur. Deux signaux $\overline{CS}$ (Chip Select) sont associés à la RAM; il s'agit d'une part, du signal transitant par la ligne K6 connectée à la broche $\overline{CS2}$ et provenant du décodeur d'adresses, et d'autre part, du signal acheminé par la ligne A9 connectée à la broche CS1.
Pour réaliser la sélection entre les ports A et B, ainsi que le sens du transfert des données, on se sert des lignes AØ, A1 et A3. Notre exposé sur la structure du PIA est nécessairement succinct, mais, nous réserverons la totalité du chapitre 5 à la description du software qui lui est associé.

Les contacts avec l'extérieur

Le clavier et l'affichage sont les organes du contact tactile et visuel, utilisés par le monde extérieur au Junior Computer. Ils constituent le tableau de bord de cette "boîte noire" à laquelle nous faisions allusion au début de ce chapitre. En fait, ils peuvent même être assimilés à une part du monde extérieur, incorporée dans le Junior Computer. Le cordon de liaison se compose de quatre fils connecteurs venant du port B, de sept fils venant du port A et de deux fils en relation avec le bus de commande. Ces derniers acheminent les signaux RESET et STOP, déjà mentionnés, et qui résultent d'une pression sur les touches correspondantes, respectivement RES et ST. Grâce au commutateur S24 (touche STEP), il est possible de choisir entre le déroulement normal du programme et le déroulement pas à pas.
Les touches que nous n'avons pas encore citées sont représentées sous la forme de commutateurs (S3 . . . S23) disposés selon une matrice rectangulaire comportant trois lignes horizontales et sept colonnes verticales, à la partie inférieure de la figure 4. Seize touches sont réservées à l'inscription de données au format hexadécimal (le chapitre 2 donnera toutes indications sur l'écriture hexadécimale). Dans le cas présent, il nous faut prendre l'expression "données" dans son sens le plus large, car les informations relatives aux adresses en font aussi partie. Les cinq touches restantes sont toutes affectées à des fonctions de commande, dont le chapitre 3, consacré au programme d'initiation, traitera en détail.
Les données destinées à l'affichage comme celles venant du clavier sont acheminées depuis le port A par sept fils de liaison; on a heureusement tiré

parti du fait qu'un port peut être configuré soit en entrée, soit en sortie. Des signaux sont appliqués périodiquement aux afficheurs Di1 . . . Di6, par l'intermédiaire du programme moniteur. Di1 . . . Di4 affichent une adresse au format hexadécimal, Di5 et Di6 définissent la donnée correspondant à l'adresse, également en écriture hexadécimale. Le même software du programme moniteur interroge périodiquement les 21 touches pour savoir si l'une d'entre elles a été pressée, et, dans l'affirmative, laquelle. Le programme, ayant détecté la touche, se charge d'exécuter la fonction correspondante. L'effet de rebondissement des contacts, résultant d'une succession de fermetures et d'ouvertures à un rythme très rapide, pendant un certain temps après que la touche ait été pressée, est ignoré grâce au programme.

L'élément assurant la périodicité du trafic des informations est logé dans IC7. Ce circuit intégré comporte 10 sorties qui sont portées tour à tour et périodiquement à l'état bas ("Ø") en fonction des bits à "1" et à "Ø" présents aux broches de connexion des quatre lignes PB1 . . . PB4 du port B. Les sorties sont connectées aux lignes de commutateurs S3 . . . S23 ainsi qu'aux six cathodes communes des afficheurs Di1 . . . Di6. Par conséquent, l'une des sorties de IC7 reste inemployée. Si à un moment donné, une cathode déterminée est portée à l'état "Ø" (ou tension nulle), parce que la sortie correspondante de IC7 a été sélectionnée, un caractère pourra être reproduit par l'illumination de un ou plusieurs des sept segments non encore allumés de l'afficheur concerné. Un segment s'illumine lorsque

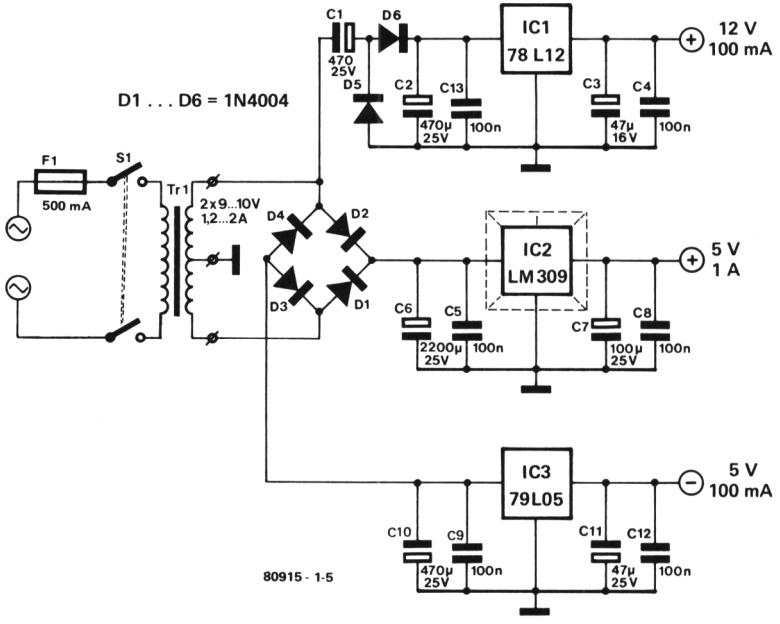


Figure 5. Circuit d'alimentation du Junior Computer délivrant trois tensions stabilisées.

la sortie du buffer (IC11) à laquelle il est connecté n'est pas à l'état bas. L'état de cette sortie est à son tour déterminé par la présence de "1" et de "Ø" sur les lignes PAØ . . . PA6, cette configuration étant elle-même gouvernée par le software.
Si l'une des trois lignes de commutateurs S3 . . . S23 est sélectionnée, le programme reconnaît la touche de cette ligne qui a été pressée. Tout bien considéré, ce mode de communication par *multiplexage*, grâce au clavier et à l'affichage, est une méthode alliant l'élégance de la solution à l'économie de matériel. Une réduction des dépenses en hardware permet une plus grande libéralité à l'égard du software, si l'on se souvient qu'un emplacement mémoire coûte moins cher que la même surface garnie de composants intégrés.
Reste S25 grâce auquel l'affichage peut être mis hors service lorsqu'il n'est pas nécessaire. Par exemple, lorsqu'on veut communiquer de l'extérieur avec le Junior Computer, par l'intermédiaire du connecteur I/O, le scintillement inutile des afficheurs sera éliminé de cette manière.

L'alimentation

Le circuit d'alimentation est présenté en figure 5. C'est un montage d'un type standard délivrant trois niveaux de tensions, à savoir + 5 V pour tous les circuits intégrés et les afficheurs, + 12 V et - 5 V pour l'EPROM IC12. Le découplage des circuits intégrés est assuré par les condensateurs au tantale C5 . . . C13 visibles sur la figure 4. L'alimentation ne devrait susciter aucun problème.

Ainsi s'achève la description du schéma du Junior Computer. Nous avons tenté de ne pas en rester à l'aspect superficiel des choses, tout en évitant de nous noyer dans les détails. Nous en venons maintenant à la réalisation du Junior Computer.

La réalisation du Junior Computer

Trois phases principales

Il est certain que le Junior Computer ne sera pas assemblé par quelque coup de baquette magique, mais, de là à prétendre que l'entreprise soit très difficile, c'est évidemment exagéré. Par contre, il est indispensable que vous lisiez très attentivement l'exposé qui va suivre, avant d'entreprendre la moindre action.
La procédure de construction comporte trois phases principales. D'abord, les trois platines des circuits imprimés doivent être garnies de leurs composants. De ce point de vue, il est possible que l'un ou l'autre d'entre nos lecteurs ait résolu de réaliser lui-même les circuits imprimés. A l'amateur inexpérimenté, nous suggérons de prendre connaissance d'un article intitulé "Circuits imprimés et soudage", paru dans le numéro 4 de Nov/Déc 1978 d'Elektor, préalablement à toute décision. En tout état de cause, vous pourrez vous procurer les circuits imprimés percés et prêts à recevoir leurs composants; ils sont également disponibles chez de nom-

breux revendeurs de composants. Après l'achèvement de la mise en place des composants sur les platines et le montage de quelques pontages les interconnectant, suit la deuxième étape qui est une phase de test, laquelle peut être terminée assez rapidement, surtout si l'on s'est attaché à choisir des composants de bonne qualité. Enfin, la troisième phase demandera un peu d'habileté manuelle pour la mise en place du Junior Computer dans son coffret, ce qui devrait se faire sans problème.

Le circuit imprimé

Le junior Computer est ce qu'on appelle parfois un "single board Computer", ce qui veut dire "ordinateur à circuit imprimé unique", ou encore, que tous les composants électroniques sont montés sur une seule platine. La vérité nous oblige à préciser que, associés à la platine principale, existent deux petits circuits imprimés, l'un pour l'alimentation et l'autre pour l'affichage. En fait, les afficheurs auraient pu très bien être partie intégrante du circuit principal. Mais la taille de ce dernier eut été alors trop importante. Des raisons d'ordre pratique nous ont conduits à incliner les afficheurs selon un angle de 45° par rapport à la platine principale, la plus évidente étant l'augmentation de leur lisibilité. Convenons donc que nous avons bien un Junior Computer "single board".
Nous sommes très près d'en venir à la soudure des composants, mais auparavant nous voudrions ajouter quelques précisions sur le circuit principal. C'est une platine double face, ce qui signifie que des pistes cuivrées sont gravées des deux côtés de la surface portante, et, que, dans ce cas, des composants seront soudés sur chaque face. Certaines liaisons électriques existent entre des pistes situées sur des faces opposées et elles sont réalisées grâce à des trous dont la paroi est métallisée par le dépôt d'une couche de métal conducteur afin d'assurer la continuité du circuit.

Le circuit principal

Le constructeur amateur qui aura décidé de réaliser lui-même [1] la platine, devra effectuer les liaisons par trous métallisés; leur nombre est supérieur à 600 . . . L'utilisation du circuit EPS 80089-1 supprime cette obligation. Précaution élémentaire, il est nécessaire de contrôler que tous les trous métallisés assurent une bonne liaison électrique entre les deux faces du circuit, ce qui peut se faire à l'aide d'un contrôleur universel connecté en ohmmètre. On peut aussi utiliser un contrôle acoustique en se servant de la méthode illustrée par la figure 14a. On connecte la sortie basse tension d'un transformateur de sonnette à l'un des câbles d'alimentation de la sonnette. Chacun des deux autres fils conducteurs est muni d'une "électrode"; ce peut être une longueur de fil de câblage rigide. Si la métallisation de la paroi du trou est parfaite, la sonnerie retentit. Cette méthode

---

[1] *Les dimensions de ce livre ne permettent pas une impression du circuit principal en vraie grandeur. Pour ceux qui veulent réaliser eux-mêmes la platine, le numéro d'avril 1980 d'Elektor le reproduit à l'échelle 1/1 dans l'article de présentation du Junior Computer.*

offre l'avantage de dispenser d'observer l'aiguille du contrôleur quelques centaines de fois, afin de s'assurer de son immobilité. Il est possible d'obvier à la défaillance d'une liaison, par la mise en place d'un fil vertical (voir figure 14b) ou par celle du fil d'extrémité d'un composant. L'immobilisation de la platine par serrage entre les mâchoires d'un étau permet de conserver la liberté d'utilisation des deux mains.
L'implantation des composants débutera par la face supérieure de la platine; la figure 6 en donne l'illustration. La face inférieure sera garnie ensuite conformément aux indications de la figure 7. Le tracé des pistes sur la face supérieure est présenté en figure 8 et celui des pistes de la face inférieure l'est en figure 9. Nous constatons que les 23 touches, 2 commutateurs à bascule et, éventuellement, le connecteur I/O à 31 broches sont disposés sur la face supérieure. Le montage des résistances, des condensateurs, des circuits intégrés, de quelques autres composants ainsi (éventuellement) que du connecteur d'extension à 64 broches se fait sur la face interne, ce qui est en opposition avec la disposition traditionnelle. Mais, du simple point de vue de l'efficacité du système électronique, il n'y a aucun inconvénient à adopter cette solution.
Cette fois, c'est bien vrai, nous allons souder. Les composants (voir la figure 7, ainsi que la liste s'y rapportant) vont être montés sur la platine dans l'ordre que voici:

1. On commence par souder (pour ceux que cela intéresse, voir notre article déjà mentionné "Circuits imprimés et soudage" Elektor Nº 4, nov/déc 1978) les résistances R1 . . . R20. Après soudure, couper immédiatement les longueurs superflues des fils aussi près que possible du point de soudure. On évite ainsi des ennuis ultérieurs, tels que le shuntage de deux points de soudure. A l'intention de ceux qui ne sont pas familiarisés avec le code des couleurs des résistances, nous en donnons un tableau ci-dessous:

330k : orange-orange-jaune-(or)
3k3 : orange-orange-rouge-(or)
4k7 : jaune-violet-rouge-(or)
330Ω : orange-orange-marron-(or)
68Ω : bleu-gris-noir-(or)
2k2 : rouge-rouge-rouge-(or)
68k : bleu-gris-orange-(or)

La résistance comporte généralement quatre anneaux colorés; le quatrième de couleur argent ou or donne l'indication de la tolérance (précision de la résistance) et donc de la mesure dans laquelle la valeur réelle peut s'écarter de la valeur nominale. Le plus souvent, vous utiliserez des résistances à 5% (or); parfois, elles seront à 1% (marron) ou 2% (rouge).

2. Suit alors le montage de la diode D1, qui exige que nous fassions très attention au respect de la polarité. Le plus souvent, pour ne pas dire toujours, le point de connexion de la cathode (sommet du triangle) est repéré par un anneau. En cas de doute, c'est le contrôleur universel qui tranchera (par la mesure de la résistance selon les deux possibilités de raccordement).
3. Les condensateurs C1, C3 et C4 sont soudés.
4. Les condensateurs électroniques C2 et C5 . . . C14 sont connectés.

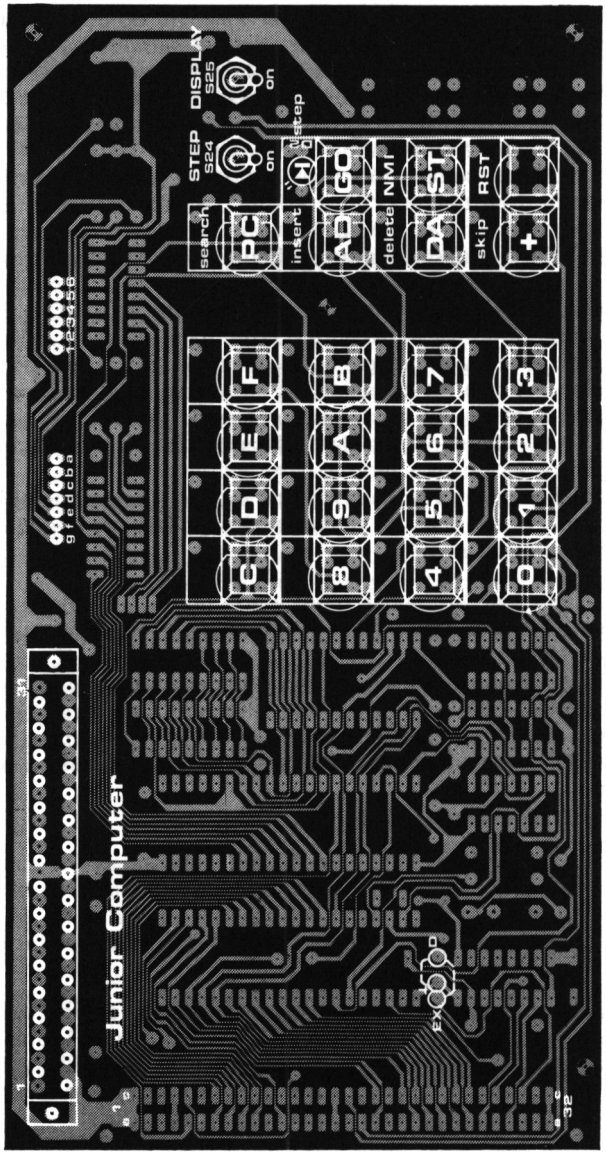


Figure 6. Implantation des composants sur la face supérieure de la platine du circuit imprimé double face principal (EPS 80089-1).

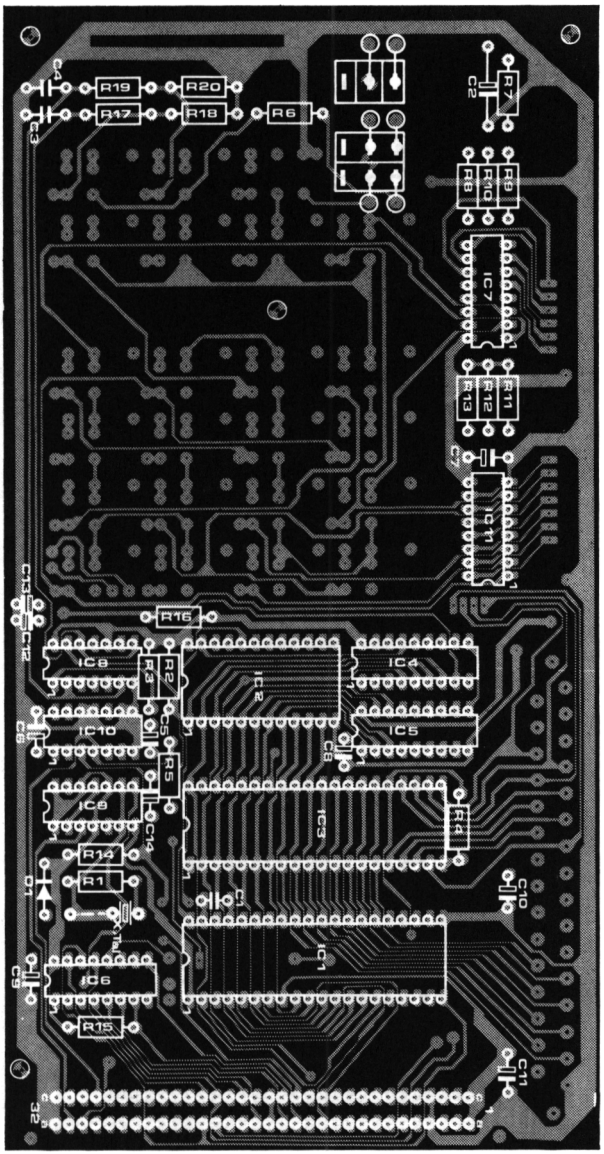


Figure 7. Liste des composants implantés sur la face inférieure de la platine principale. Toutes les résistances et les diodes, tous les condensateurs et les circuits intégrés, ainsi que le quartz y sont montés.

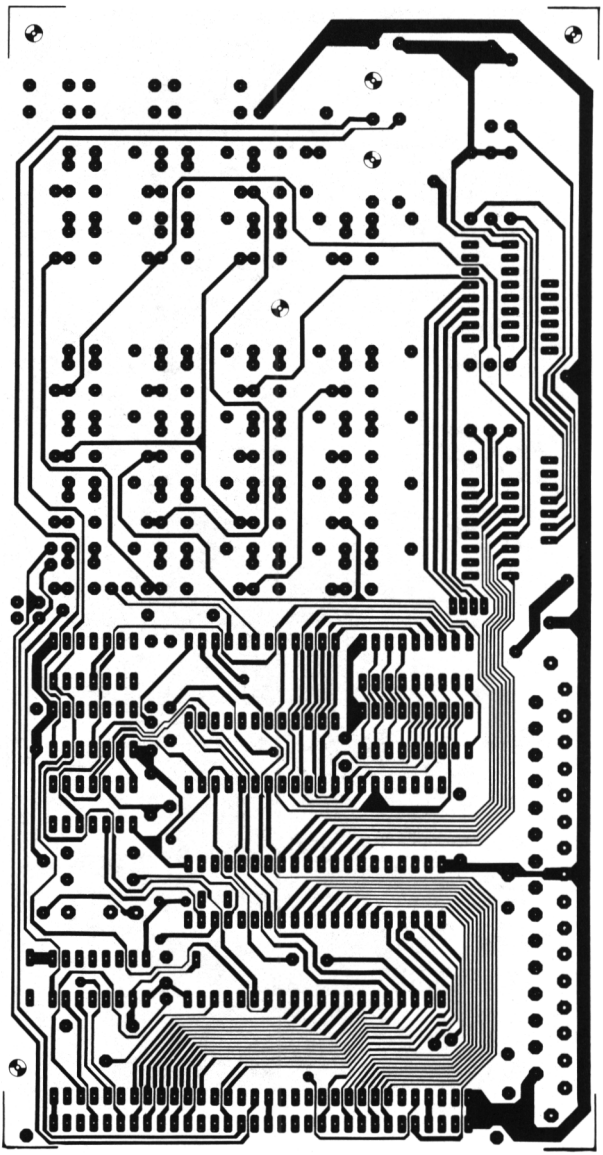


Figure 8. Tracé des pistes cuivrées à la surface supérieure du circuit principal.

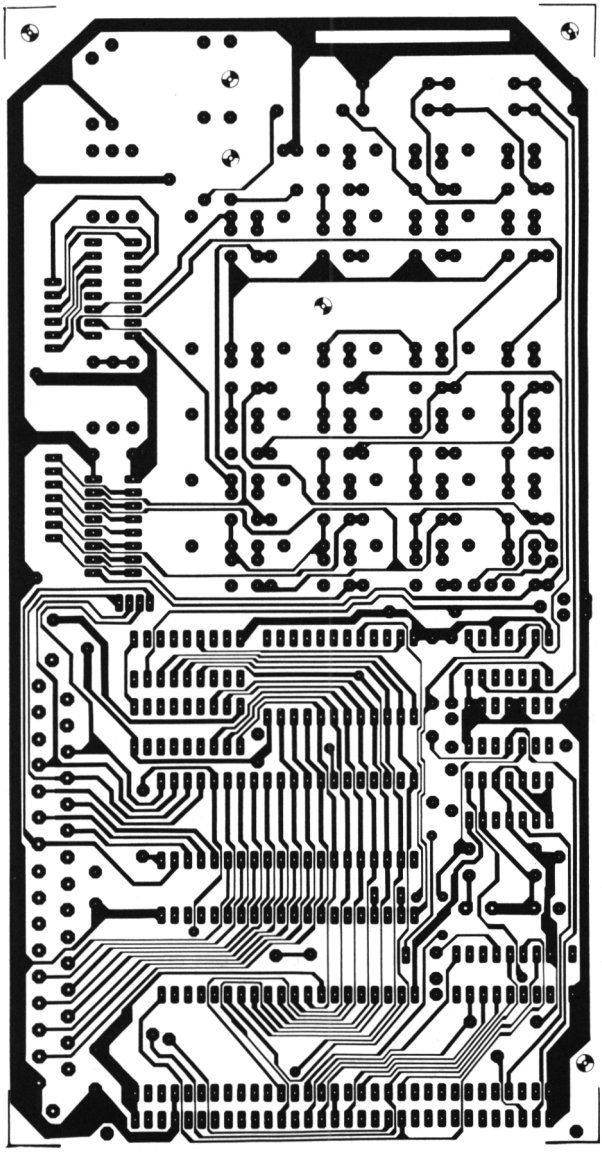


Figure 9. Tracé des pistes cuivrées à la surface inférieure du circuit principal.

Liste des composants équipant le circuit principal du Junior Computer

La figure 4 représente le schéma de principe, les figures 6 et 7 l'implantation des composants sur chacune des faces de la platine, les figures 8 et 9 le tracé des pistes cuivrées sur l'une et l'autre face.

Résistances:
R1 = 330k
R2,R3,R4,R14,R15,R16 = 3k3
R5 = 4k7
R6 = 330 Ω
R7,R8,R9,R10,
R11,R12,R13 = 68Ω
R17,R19 = 2k2
R18,R20 = 68k

Condensateurs:
C1 = 10 p céramique
C2 = 47μ/6V tantale
C3,C4 = 100n MKM ou MKH
C5,C6,C7,C8,C9,C10,C11,
C12,C13,C14 = 1μ/35 V tantale

Semiconducteurs:
IC1 = 6502 (Rockwell,MOS)
IC2 = 2708
IC3 = 6532 (Rockwell,MOS)
IC4,IC5 = 2114
IC6,IC7 = 74LS145,74145
IC8 = 556
IC9 = 74LS00,7400
IC10 = 74LS01,7401
IC11 = ULN2003 (Sprague, Motorola), XR2203 (Exar), 9667 (Fairchild)
D1 = 1N4148
D2 = LED rouge

Divers:
Quartz = 1 MHz (peut être éliminé éventuellement)
S1 . . . S23 = Digitast (Shadow), dont une avec une découpe pour LED
S24 = Commutateur bipolaire ON/OFF
S25 = Commutateur unipolaire ON/OFF
1 support IC 24 broches (pour IC2)
2 supports IC 40 broches (pour IC1 et IC3)
1 connecteur mâle 64 broches, d'équerre DIN41612 (en option)
1 connecteur femelle 31 broches DIN41617 (en option)
(s'il n'est pas fait usage du connecteur d'extension à 64 broches, il faut au moins 4 picots de Ø = 0,8 mm, pour les connexions d'alimentation)
1 Circuit imprimé EPS 80089-1

Il y a deux orientations pour monter une résistance ou un condensateur. Par exemple, les condensateurs électrolytiques ont un pôle positif et un pôle négatif. Sur le schéma d'implantation des composants, le pôle positif est représenté par une barre rectangulaire dont le périmètre délimite un espace vide (▭), tandis que le pôle négatif est figuré par une barre rectangulaire pleine (▬)

5. C'est maintenant au tour des circuits intégrés. L'usage courant est de se servir de supports soudés directement sur la platine, ce qui permet ensuite d'y embrocher les circuits. L'avantage d'un tel montage est qu'il permet de retirer un circuit de la platine en un instant et de l'y remettre ou de le remplacer par un autre, tout aussi rapidement. Par contre, l'inconvénient est qu'il exige la fiabilité de 14, 16, 24 ou 40 points de contact, selon le nombre de broches du circuit intégré. Dans le cas du Junior Computer, on a opté pour le juste milieu. Trois circuits intégrés sont pourvus d'un support: IC1 et IC3 (40 broches) et IC2 (24 broches); il va de soi que nous choisirons des supports de bonne qualité. Les autres circuits intégrés seront soudés directement sur la platine.

6. Sur le schéma d'implantation des composants, un circuit intégré est représenté par un rectangle dont l'un des petits côtés présente une découpe arrondie en demi-cercle. La broche immédiatement à gauche est repérée par le chiffre "1". Ces deux repères figurent également sur le boîtier d'un IC, et il faut donc faire coïncider les uns et les autres. Ne vous fiez *jamais* au texte imprimé par le fabricant sur le boîtier pour déterminer l'orientation de ce dernier.
S'agissant de l'EPROM IC2, l'utilisation d'un support autorisera l'échange facile du programme moniteur du Junior Computer.
7. Le quartz de 1 MHz est monté, bien que, si vous le vouliez, vous puissiez renoncer à son utilisation, car elle n'est pas de stricte nécessité. Le quartz présente l'avantage de délivrer une fréquence d'horloge d'une extrême précision. Mais, dans la plupart des cas, le désastre n'est pas grand s'il faut attendre quelques microsecondes de plus (6 ou 6,05) avant qu'une instruction soit exécutée.
8. Vient alors le montage du connecteur d'extension à 64 broches, qui, lui non plus n'est pas indispensable. Si vous n'avez pas encore de projets d'extension, ce qui est très vraisemblable lors d'un premier tour d'horizon, vous pouvez vous dispenser, provisoirement, de l'installer. Cependant, dans un tel cas, il faut revoir la distribution des lignes d'alimentation. Les points d'alimentation sont:

+ 5V : broches 1a ou 1c
masse : broches 4a, 4c, 32a ou 32c
- 5V : broche 18a
+ 12V : broche 17c

Lorsqu'on utilise un connecteur, les points de connexion cités ci-dessus peuvent être facilement retrouvés puisque les numéros restent affectés aux connexions. Si l'on renonce au connecteur à 64 broches, il est conseillé de se servir de cinq picots fixés aux points de connexion correspondants. Sur la platine, sont gravés les numéros les plus extrêmes 1a/1c et 32a/32c. Une connexion devant être obligatoirement à sa bonne place, il est nécessaire de compter soigneusement les petits trous, afin que l'ordre exact soit respecté. Une erreur dans ce domaine serait très préjudiciable au Junior Computer.

Tous les composants de la face inférieure de la platine étant montés, nous passons au montage de ceux de la face supérieure.

9. Le seul pontage de la platine est soudé entre les points ⊥ et D.
10. Les deux commutateurs à bascule sont fixés sur la platine, le boîtier étant positionné du côté de la face inférieure. 6 longueurs de fil de câblage relient les bornes des commutateurs au circuit imprimé, côté face inférieure. Elles sont disposées très nettement à la surface de la platine.
11. Le connecteur à 31 broches est ensuite installé.
12. Restent les 23 touches et la diode D2. Cette phase de l'assemblage doit être exécutée avec un soin tout particulier, car, au niveau des touches, une poussée mécanique assez forte est exercée sur la platine. Au montage de chaque touche, assurez-vous qu'il existe un bon contact entre le boîtier de la touche et la platine. La diode D2 est montée avant la

touche GO; ce faisant, veillez très scrupuleusement au respect de la polarité. La connexion de cathode de D2 jouxte le côté de la platine.

Provisoirement, nous en avons terminé avec la platine principale. Quoique, avant de nous tourner vers d'autres tâches, il serait bon que nous nous assurions que des résidus de soudure ne risquent pas d'établir des connexions électriques indésirables. La position correcte des circuits intégrés et des condensateurs électrolytiques mérite également une dernière vérification.

Le circuit d'affichage

Nous ne devrions pas lui consacrer un temps considérable, car il est relativement simple. La figure 10 montre le plan d'implantation des composants et la figure 11 le tracé des pistes. En fait, le montage est scindé en

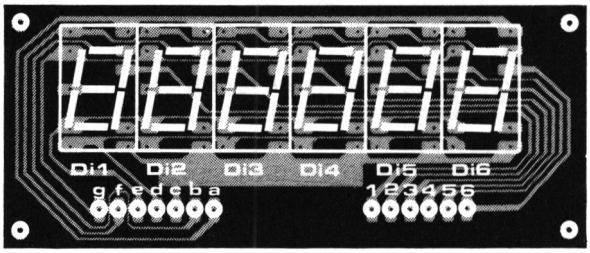


Figure 10. Implantation des composants équipant le circuit imprimé d'affichage (EPS 80089-2).

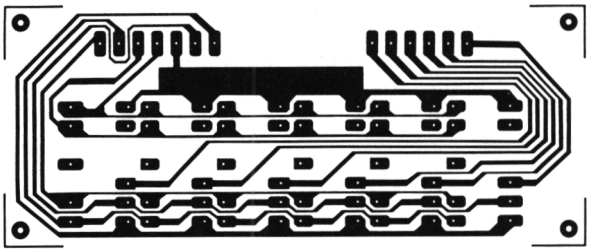


Figure 11. Tracé des pistes cuivrées du circuit d'affichage.

Liste des composants équipant le circuit d'affichage du Junior Computer

La figure 10 représente l'implantation des composants et la figure 11 le tracé des pistes cuivrées.

Semiconducteurs:
Di1,Di2,Di3,Di4,Di5,
Di6 = MAN4640A (cathode commune) Monsanto

Divers:
1 circuit imprimé EPS 80089-2

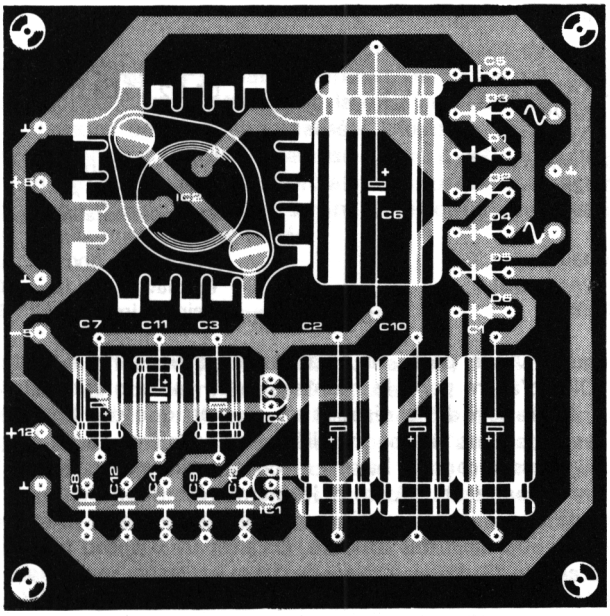


Figure 12. Implantation des composants équipant le circuit d'alimentation (EPS 80089-3).

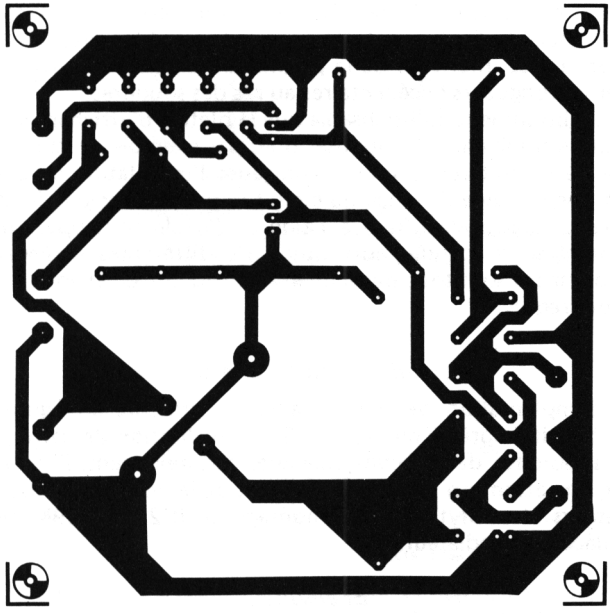


Figure 13. Tracé des pistes cuivrées du circuit d'alimentation.

Liste des composants équipant le circuit d'alimentation du Junior Computer

Condensateurs:
C1,C2,C10 = 470μ/25 V
C3,C11 = 47μ/25 V
C4,C5,C8,C9,C12,
C13 = 100n MKM ou MKH
C6 = 2200μ/25 V
C7 = 100μ/25 V

Semiconducteurs:
IC1 = 78L12ACP (5%)
IC2 = LM309K
IC3 = 79L05ACP (5%)
D1,D2,D3,D4,D5,D6 = 1N4004

Divers:
Tr1 = transformateur 2 x 9 à 10 V/1,2 à 2 A
S1 = commutateur bipolaire ON/OFF
F1 = fusible 500 mA, avec socle
1 radiateur plat (pour IC2)
1 circuit imprimé EPS 80089-3

deux phases. D'une part, la fixation des afficheurs à sept segments Di1 . . . Di6; d'autre part, le montage de la petite platine sur la platine principale. En raison de la disposition asymétrique des broches de connexions, les afficheurs seront installés sans problème. Sept longueurs de fil de câblage assureront la liaison électrique des points a . . . g, tandis que six autres longueurs joueront le même rôle pour les points 1 . . . 6. Les points de connexion se distinguent très nettement à la surface du circuit principal, au-dessus des touches. On prépare d'abord treize longueurs de 1,5 à 2 cm de fil d'une section de 1mm et on les soude dans les treize trous restés libres à la surface de la platine d'affichage, côté cuivré. Si les extrémités des fils dépassent, côté composants, on les coupe au ras des soudures. Les treize pontages doivent être pliés et orientés de telle manière qu'ils s'enfilent tous dans les trous de connexion de la platine principale. La platine d'affichage doit faire un angle de 45° avec le circuit imprimé principal. Tout ayant été bien préparé, les treize pontages sont soudés sur la face inférieure de la platine principale. Nous n'oublions pas de couper les longueurs excédentaires au ras des soudures.
Il n'est pas absolument indispensable que la platine d'affichage soit montée sur la platine principale. A supposer que, par exemple, le choix d'un certain type de coffret rende plus avantageuse l'installation du circuit d'affichage à une certaine distance du circuit imprimé principal, il est possible de réaliser la liaison électrique à l'aide de "câble en nappe" formé d'un certain nombre de conducteurs isolés, de différentes couleurs. Il faut alors accorder un soin tout particulier à la soudure de chaque conducteur dans l'ordre requis.

Le circuit imprimé d'alimentation

Le plan d'implantation des composants est présenté en figure 12 et le tracé du circuit imprimé en figure 13. La réalisation de cette platine ne devrait pas susciter de difficulté majeure. Naturellement, il est nécessaire de porter une attention spéciale au respect de la polarité des diodes et des condensateurs électrolytiques. Le montage de IC2 (LM309K) requiert la mise en place d'un radiateur.

Et ensuite...

Les trois circuits imprimés sont donc complètement équipés. Il faut encore

réaliser le raccordement du transformateur d'alimentation Tr1 au circuit d'alimentation, ainsi qu'au secteur, par l'intermédiaire de l'interrupteur secteur S1 et du fusible F1. Mais, attention, ne mettez pas la fiche dans la prise de courant! Dans l'attente de la mise en place définitive dans le coffret, les liaisons peuvent être réalisées provisoirement en se gardant de ne pas leur conférer la solidité requise. La vérification du bon état des connexions d'alimentation est primordiale à un fonctionnement de longue durée du Junior Computer. Elles seront fixées définitivement après que l'alimentation elle-même ait été testée.

Premiers essais

Nous arrivons au moment passionnant qui va nous permettre de tester le bon fonctionnement de notre Junior Computer.
En premier lieu, vous comprendrez qu'un contrôle final des trois circuits imprimés est nécessaire. Rien ne vous empêche d'avoir recours aux bons offices d'un de vos amis, amateur, comme vous, d'un travail bien fait. Ensuite, la fiche est enfoncée dans le socle de la prise de courant. *Attention, l'alimentation n'est pas encore appliquée au circuit principal!* Cette opération se fait à l'aide de S1 et nous vérifions au contrôleur universel (connecté en voltmètre pour tension continue) la valeur des trois

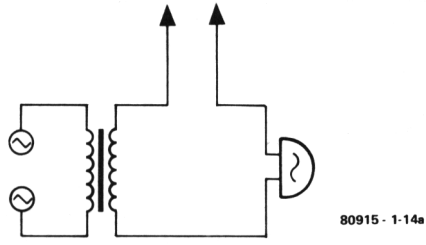


Figure 14a. Il n'est pas indispensable que le contrôle des connexions électriques soit réalisé à l'aide d'un contrôleur universel (connecté en ohmmètre). Il peut être fait également avec un montage regroupant une sonnette et son transformateur basse tension, délivrant un signal acoustique.

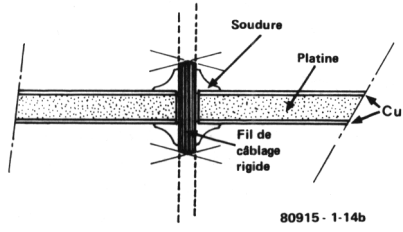


Figure 14b. Illustration de la réalisation d'une liaison verticale par soudure d'une longueur de fil de montage rigide au travers d'un trou de la platine, dans le cas d'un circuit imprimé gravé par le constructeur amateur.

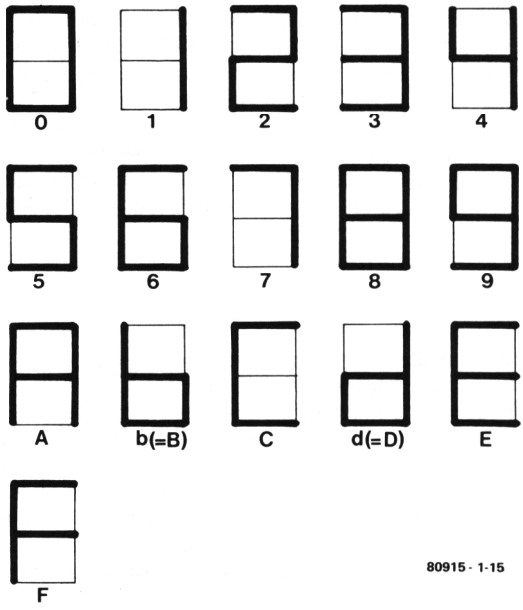


Figure 15. Le bon fonctionnement du Junior Computer est attesté par l'apparition de six caractères hexadécimaux sur les afficheurs, après qu'une pression ait été exercée sur la touche RST. La figure donne l'illustration des seize caractères susceptibles d'apparaître.

tensions d'alimentation par rapport à la masse. Elles doivent être égales à ± 5% près à + 5, - 5 et + 12 V. Si ce n'était pas le cas, il faudrait vérifier complètement le circuit d'alimentation séparément. Mais, ce type de circuit ne devrait pas engendrer de problèmes majeurs.
Tout étant en ordre, on le met hors tension et les quatre pontages entre l'alimentation et la platine principale sont mis en place. A cette occasion, nous réitérons notre conseil du point 8 de la liste des opérations d'assemblage du circuit principal: faites très attention à respecter l'ordre exact de disposition des connecteurs!
Ceci fait, nous plaçons le commutateur STEP (S24) en position OFF et le commutateur DISPLAY (S25) en position ON. Le Junior Computer est mis sous tension à nouveau, et...
l'affichage reste totalement obscur. Ne vous affolez pas! C'est normal. Il faut commencer par presser la touche RST. Si tout marche bien, chaque afficheur fait apparaître un caractère hexadécimal quelconque. Ainsi que nous l'avons déjà dit, le chapitre 2 vous expliquera ce qu'est l'écriture hexadécimale. Pour l'instant, il suffit de savoir que la combinaison des segments illuminés peut se présenter selon le schéma qu'illustre la figure 15.
En ce point, nous anticipons sur la progression de notre lecture et nous imaginons que nous avons déjà pris connaissance du chapitre 3, où il est question d'une instruction de saut dite conditionnelle, laquelle se formule en l'occurrence de la manière suivante: si tous les afficheurs sont illuminés,

vous pouvez sauter la lecture du paragraphe suivant et lire tout de suite celui qui traite du montage du Junior Computer dans son coffret. Nous nous retrouverons dans quelque temps. Pour ceux qui n'ont pas obtenu d'emblée le résultat escompté, et nous sommes persuadés qu'ils sont peu nombreux, nous restons en leur compagnie afin de les aider à pratiquer telle ou telle rectification mineure.

Quelques difficultés... et leur remède

Supposons que l'alimentation fonctionne, que peut-il se passer?

- On peut penser à quelque cause très générale, comme, par exemple, un excédent de soudure, qui n'est pas toujours fort apparent. De petites particules de soudure inférieures à 1mm en diamètre sont peut-être la cause de contacts inopportuns.
- Des soudures peuvent être défectueuses ("soudure sèche"), il ne reste plus qu'à réchauffer les points douteux au fer à souder, en les garnissant bien d'étain.
- Contacts défectueux entre les connecteurs mâle et femelle ainsi qu'entre les broches des circuits intégrés et leurs supports. Une vérification approfondie des points de contact s'impose. Eventuellement, pratiquer un nettoyage à l'alcool des éléments concernés. Préalablement, il n'est pas inutile de s'assurer du bon contact mécanique entre les parties mâle et femelle, en enfonçant bien les parties mobiles.
- En ce qui concerne les platines fabriquées par l'amateur, il se pourrait que la piste cuivrée d'un circuit présente un défaut à la suite d'une gravure incomplète ou de la présence d'une particule ou d'un cheveu. Il faut contrôler les tracés à la loupe et faire disparaître d'éventuels shunts de cuivre à l'aide d'une lame de canif; bien veiller à ne pas endommager les pistes!

Vérifier un point déjà évoqué: le respect de la polarité des diodes et des condensateurs électrolytiques, ainsi que le montage correct des circuits intégrés sur leurs supports. Rien n'exclut que le raccordement des pontages reliant l'alimentation au circuit principal ait été l'occasion d'une erreur; c'est à contrôler.

C'en est fini des considérations d'ordre général. Voyons maintenant quelques petits "trucs" spéciaux:

- à l'aide d'un contrôleur pour tension continue (calibre 6, 10 ou 12 V), mesurer la tension entre les points 7 et 13 de IC8; elle doit être d'environ 5 V et retomber à 0,5 V après une pression sur la touche RST. Si ce n'est pas le cas, on peut incriminer IC8 (le timer double), R2 ou la touche RST elle-même.
- si rien n'est anormal dans ce qui précède, faire un contrôle à l'ohmmètre et s'assurer que la broche 12 de IC6 est à la masse. Si ce n'est pas le cas, il se pourrait que le seul pontage du circuit soit mal monté.
- il est possible également de tester séparément le fonctionnement du générateur d'horloge. A l'aide d'un oscilloscope, observons les signaux présents aux points 30a et 27a du connecteur d'extension. Si tout va bien, nous décelons les trains d'impulsions Ø1 et Ø2 avec une fréquence de répétition de 1MHz et une amplitude de 3 à 5 V. S'il s'agit d'un appareil à double trace, il faut s'assurer que les deux impul-

sions ne se chevauchent pas (c'est-à-dire que leur tension ne passe pas simultanément par la valeur de crête, 3 à 5 V). A supposer que l'écran reste vide, l'horloge ne fonctionne pas. C1, IC9 et D1 peuvent être en cause.

Si malgré toutes les mesures générales ou spécifiques citées ci-dessus, un problème persistait, nos lecteurs ont la faculté de poser des questions techniques à la rédaction d'Elektor, soit par écrit, soit par téléphone le lundi entre 13h30 et 16h30.

Le coffret

Le point important, de ce point de vue, c'est que le coffret doit permettre de protéger efficacement le précieux équipement interne contre les influences extérieures. Cette condition étant remplie, il serait bon que le choix se fasse en gardant présente à l'esprit la nécessité de pouvoir réaliser quelques extensions ultérieures. Car le besoin ne manquera pas de s'en faire sentir. En général, on peut envisager deux types principaux de coffret: soit le genre "boîte à cigares", soit le modèle de "bureau". Concernant ce dernier type, si le circuit d'affichage est installé sur la paroi frontale verticale, la commodité d'utilisation du Junior Computer s'en trouvera renforcée.

Il y a lieu en tout cas de prévoir les découpes ou les ouvertures pour les afficheurs, le clavier, le support du fusible, l'interrupteur secteur et le cordon d'alimentation. En plus des quatre trous de fixation habituels, la platine principale en comporte un cinquième au voisinage des touches, destiné à accroître la résistance au fléchissement. Des entretoises permettent de régler l'écartement.

La finition demande que l'on choisisse des touches munies d'inscriptions durables et nettes, harmonisées à l'ensemble (la figure 6 donne des informations utiles à ce sujet), et que l'on puisse bien distinguer l'une de l'autre.

Quelques remarques finales

Au sortir de fabrication, IC2, l'EPROM n'est évidemment pas programmée et ne peut donc, telle quelle, être utilisée dans le Junior Computer; il va falloir préalablement y inscrire 1024 x 8 bits (des "1" et des "Ø") et qu'ainsi soit composé le programme moniteur. Le listing de ce dernier (hexdump) figure à l'appendice 3, afin que le lecteur disposant d'un programmateur de PROM (ainsi que d'un "personality module" type 2708) puisse programmer lui-même le circuit intégré. Néanmoins, d'ici peu de temps, beaucoup de revendeurs de composants disposeront d'EPROM "hexdump". L'expérience a prouvé que, sans une EPROM bien programmée (IC2) le Junior Computer refuse de fonctionner et que le test fondamental de l'illumination des 6 caractères hexadécimaux est voué à l'échec, ce qui pourrait avoir pour conséquence que l'on se lance dans la recherche d'une défectuosité du matériel (hardware) qui n'existerait pas!!

2

Systèmes numériques

codes et opérations numériques

C'est probablement au fait que l'être humain possède dix doigts que nous sommes redevables du système décimal. Par suite, nous effectuons nos calculs à l'aide de nombres décimaux que nous employons également dans le langage (secret) de nos codes. Le Junior Computer doit, lui aussi, effectuer des opérations arithmétiques et traiter toutes sortes de données codées. Comparativement à l'homme, ses moyens sont plus réduits car son système de numération ne comporte que deux chiffres, d'où l'expression "système binaire".
C'est à celui-ci qu'est consacré le présent chapitre.

Prenons un 1, un 9, un 8 et un 1 et écrivons-les l'un à la suite de l'autre, de la gauche vers la droite:

1981

Evidemment, cela nous fait immédiatement penser à un nombre. Par exemple à un millésime, ou encore au prix d'un appareil.
Mais la signification peut être plus subtile. Elle peut faire songer au code (non explicite) désignant "l'année prochaine". Voire même, un numéro de téléphone. Ce dernier est également l'expression d'une information codée, qui, associée à l'indicatif de la zone, détermine les états de tout un ensemble de relais et de commutateurs, ce qui permet à l'abonné d'appeler un correspondant, ou au contraire d'être appelé par un autre correspondant.
Imaginons que ce numéro de téléphone, 1981, soit au nombre des données introduites dans un ordinateur. Il n'en reste pas moins que les opérations internes de ce dernier porteront sur toute autre chose que le nombre 1981. Ce sera précisément:

1111Ø111101

Il est clair que c'est tout à fait différent. Les zéros sont barrés d'une ligne oblique. La barre oblique sert à les différencier de la lettre majuscule O. Il apparaît immédiatement que l'on ne doit pas assimiler ce groupement de uns et de zéros au nombre décimal "classique" 11.110.111.101. Il est d'ailleurs tout à fait remarquable qu'il ne soit composé que de uns et de zéros. S'agissant d'un nombre décimal de 11 chiffres, la chance que l'on ait à traiter un nombre composé exclusivement de uns et de zéros est extrêmement faible. Nous avons même calculé pour vous qu'elle est de l'ordre de deux millionièmes pour cent. Très petite, en réalité.
Tenons donc cette éventualité pour quasiment nulle. En ce qui concerne les nombres de 11 chiffres constitués de uns et de zéros, il existe $2x2x2x2x2x2x2x2x2x2x2 = 2^{11}$ combinaisons différentes susceptibles d'être réalisées. Comparativement, les dix chiffres 0 . . . 9 du système décimal permettent d'obtenir $10x10x10x10x10x10x10x10x10x10x10 = 10^{11}$ combinaisons différentes correspondant à autant de nombres à 11 chiffres "classiques".

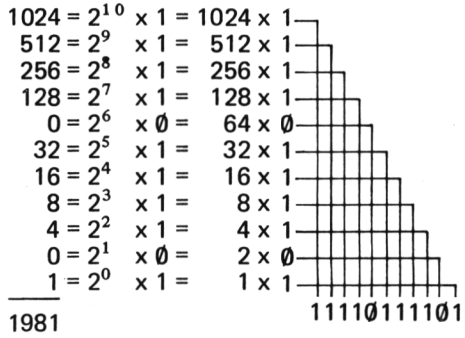
Composition d'un nombre

Tout nombre peut être considéré comme étant le résultat de l'addition de plusieurs autres nombres. De nombreuses combinaisons sont possibles, et cela d'autant plus que le nombre est plus grand. C'est exactement la même chose pour les nombres qui, additionnés l'un à l'autre, reproduiront le nombre final. On conçoit très facilement qu'il faut qu'il existe une convention pour que l'agencement des différents composants délivre le nombre final. Cette convention n'est autre que le *système de numération*. De ce point de vue, il y en a un que vous connaissez très bien, même si vous n'en êtes peut-être pas très conscient. C'est le système décimal, ou système à dix chiffres. Grâce à lui, un nombre est composé, selon son ordre de grandeur, d'unités, de dizaines, de centaines, de milliers etc. A chaque fois, il faut grouper dix unités d'un certain ordre pour obtenir une unité de l'ordre immédiatement supérieur. Notre système décimal est donc un système de *base dix*. Les unités, les centaines, les milliers etc, sont des puissances de dix. Chacune des unités d'un certain ordre doit être multipliée un nombre déterminé de fois par 0, 1, 2, 3, 4, 5, 6, 7, 8 ou 9 pour que l'addition de tous les résultats partiels permette de reproduire le nombre final. Les chiffres représentant les unités d'un certain ordre sont écrits de la droite vers la gauche selon un ordre croissant. Ainsi, le nombre 1981 se compose de:

1 unité du premier ordre
8 unités du second ordre
9 unités du troisième ordre
1 unité du quatrième ordre

$1 \times 10^0 = 1$
$8 \times 10^1 = 80$
$9 \times 10^2 = 900$
$1 \times 10^3 = 1000$
1981

On peut imaginer une autre manière de reproduire le nombre 1981:



Chaque 1 est l'expression d'une puissance de 2

Tout Ø est l'absence d'expression d'une puissance de 2

Vous constatez que maintenant, nous avons décomposé 1981 en nombres qui ne sont plus des équivalents de certaines puissances de 10, mais, de certaines puissances de 2. Dans le système décimal, chaque chiffre composant le nombre 1981 (ou tout autre) peut être remplacé par une certaine puissance de dix multipliée par l'un des symboles du système (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Par contre, lorsque la décomposition se fait à partir de puissance de 2 (2; 2x2; 2x2x2; 2x2x2x2; etc.), il n'y a plus que deux symboles utilisables, à savoir: Ø et 1. L'écriture de l'équivalent de 1981 dans le nouveau système se fait également en partant de la droite vers la gauche, et dans un ordre croissant. Vous ne serez sans doute pas étonné que nous nous soyons servi de l'exemple ci-dessus pour illustrer la conversion d'un nombre du système décimal au *système binaire*. Ne nous y trompons pas, il s'agit toujours du même nombre, 1981, mais représenté dans un autre système.

Les raisons d'un choix

On peut se demander s'il est tellement nécessaire de passer du système décimal, qui, à bien des égards, a largement fait ses preuves, au système binaire. A cette interrogation, deux réponses peuvent être formulées: oui (1) et non (Ø). En définitive, le "oui" présente quelques avantages spécifiques.

Examinons d'un peu plus près l'avantage qu'il y a à représenter le nombre décimal 1981 par son équivalent binaire 11110111101 composé de zéros et de uns. Nous avons constaté, en décomposant 1981 en nombres représentatifs de puissances de 2, que, selon le cas, il "existe" ou "n'existe pas" de puissance de 2. Et ces expressions "existe" et "n'existe pas" ont beaucoup d'analogie avec les réponses "oui" et "non". Un oui s'exprime par un 1, un non par un Ø, selon la position correspondante du chiffre.

Dans un ordinateur, les nombres et les données codées sous forme de nombres doivent être, de manière ou d'autre, consignés dans un ensemble électronique purement matériel. A chaque chiffre d'un nombre, correspond

une section d'un circuit électronique. A partir du moment où l'ordinateur voit ses opérations internes régies à l'aide d'un système de numération binaire, la représentation de tout chiffre se réduit au choix entre deux états. Et, il est tout naturel que la section de circuit électronique que nous mentionnions précédemment puisse prendre, elle aussi, deux états. Différents types de bascules conviennent à l'obtention de ces deux états que l'on appelle également niveaux logiques, et, très précisément, 1 = un logique, ou encore niveau "haut" correspondant à la présence d'une tension, et, Ø = zéro logique, ou encore niveau "bas" correspondant à l'absence de tension (0 volt).
Ce que nous venons de définir est la "logique positive" dans laquelle le niveau logique "oui" est déterminé par la tension la plus élevée; le niveau logique "non" est caractérisé par la tension la plus voisine de zéro volt. Mais il existe aussi une logique négative dans laquelle la définition est inverse.
L'avantage d'utiliser une logique binaire interne réside dans le fait que l'équipement électronique est très simple. Il est bien plus facile de réaliser quatre bascules pouvant prendre chacune deux états distincts (ce qui donne un total de seize combinaisons possibles), qu'un seul élément devant prendre dix états différents, ce qu'il faudrait faire s'il fallait reproduire un chiffre décimal. Bien entendu, cela reste du domaine du possible, mais non sans complexité.
A cela s'ajoute un autre grand avantage du binaire. Dans un paragraphe ultérieur de ce chapitre, nous aborderons l'étude de l'ordinogramme, lequel implique souvent la prise de décisions entraînant l'exécution d'opérations en fonction de l'existence de tel ou tel critère. Imaginons, qu'à un moment donné, il faille prendre une décision aux termes de laquelle, un choix doive être fait en fonction de conditions précises, entre dix actions possibles. Il sera beaucoup plus commode de substituer à un choix entre dix possibilités, dix choix entre deux possibilités; autrement dit, de remplacer un choix complexe entre dix solutions par une succession de dix choix fondés sur dix décisions entre oui et non, grâce auxquelles les dix possibilités auront été testées.
En réalité, ce dont il s'agit, c'est du remplacement d'une "chose" complexe par un certain nombre de "choses" plus simples. Cela ne vaut pas seulement pour les décisions dont nous parlions; c'est tout aussi valable pour notre exemple chiffré du début de ce chapitre. Le nombre 1981 s'exprime à l'aide de quatre chiffres dans le système décimal, alors qu'il en faut onze dans le système binaire 1111Ø111ØØ1. Mais, dans ce dernier, il n'y a que deux symboles (ou caractères), au lieu de dix, et, par conséquent, le choix se réduit à chaque fois entre deux possibilités.
C'est précisément dans le choix du système binaire, et donc de la logique OUI/NON, que réside la formidable puissance de l'ordinateur à transformer des problèmes ou des opérations complexes en un nombre plus ou moins grand de problèmes et d'opérations absolument simples.[1]

[1] *L'objet de cet ouvrage n'est pas de décrire la logique oui/non et l'algèbre booléenne qui lui est apparentée, dans les moindres détails. Cela d'autant plus que "Digit 1", un autre ouvrage de la collection Publitronic leur est consacré en grande partie.*

Bits, octets et autres

S'agissant du système binaire, et donc de tout ce qui se passe dans un ordinateur, il paraît assez étrange de parler des chiffres d'un nombre, car, dans ce cas, celui-ci ne se compose plus de chiffres, mais de *bits* (abréviation américaine de "binary digit"). Un bit peut prendre deux valeurs, Ø ou 1, et il constitue l'unité d'information dont le contenu est le plus faible qui soit.

Souvent, les bits se présentent en groupes ou ensembles, leur nombre déterminant le format de l'information. Il est d'ailleurs fait mention fréquemment de *mots*. Tout comme les mots de ce texte se composent de lettres, les mots binaires sont formés de bits.

Mais il existe d'autres termes pour désigner des mots composés d'un nombre déterminé de bits. Par exemple, il est d'usage de désigner du nom *octets* (bytes en américain) des mots groupant 8 bits (d'autres systèmes admettent des bytes d'une longueur de 16 bits).

Un mot de quatre bits s'appelle aussi un *quartet* (nibble, en américain).

Vous vous souvenez certainement que, dans le cours du chapitre 1, il a été précisé que la capacité du bus de données du Junior Computer est de 8 bits, ce qui signifie qu'à tout instant donné un octet peut y être déposé. En ce qui concerne le bus d'adresses, sa capacité est de 16 bits, soit deux octets.

Les bits et les octets correspondants ne concernent pas exclusivement l'expression binaire d'un nombre. Ils se rapportent également à tous les autres *codes numériques*. De ceux-ci, il existe un certain nombre. Par exemple:

- le code d'un type déterminé d'instruction que l'ordinateur peut exécuter (à notre requête). Ces *codes opération* sont conçus en fonction d'un type de microprocesseur précis (dans le cas du Junior Computer, le 6502). Le chapitre 3 nous les fera connaître.
- le code à quatre bits permettant de représenter les chiffres décimaux de 0 à 9. Nous reviendrons ultérieurement sur ce *code DCB* (Décimal Codé Binaire).
- le *code adresse* qui est, en fait, un nombre binaire de 16 bits (long de deux octets) désignant, en un instant donné, l'emplacement mémoire (calculé à partir de l'emplacement mémoire zéro) concerné par l'exécution d'une instruction.
- il existe également le *code ASCII* (American Standard Code for Information Interchange) regroupant sept bits pour le codage de *symboles* ou *caractères*. Il est composé de majuscules, de minuscules, de chiffres décimaux, de signes de ponctuation et autres symboles alphanumériques inspirés de ceux du telex.
- des codes appliqués à d'autres données.

Quel que soit le code numérique, il est constitué d'un certain nombre de bits dont chacun prend la valeur 1 ou la valeur Ø.

Notation hexadécimale

Sans doute êtes-vous convaincu maintenant de l'intérêt qu'il y a à substituer aux chiffres du système décimal des suites beaucoup plus longues de 1 et de Ø. Il est certain que pour l'ordinateur les manipulations seront plus

commodes; mais l'on peut se demander à juste titre s'il en va de même pour son utilisateur. Car celui-ci doit inscrire des données et leurs adresses sous forme de séries impressionnantes de 1 et de Ø. Songez seulement au bus d'adresses et à ses 16 bits. Il est pratiquement impossible que des erreurs ne soient pas commises. Alors, est-ce sans remède?
Heureusement, il n'en est rien, parce que nous pouvons faire usage de la *notation hexadécimale* d'un nombre binaire ou d'un autre code numérique.
Le système hexadécimal comporte 16 caractères et l'on dit qu'il est de base 16. Notre démarche consiste à représenter un mot binaire au moyen de la notation propre au système numérique hexadécimal. Nous avons constaté que la conversion d'un nombre du système décimal au système binaire s'accompagne d'une forte augmentation de la quantité de signes (bits) utilisés. Pour des raisons inverses, la conversion du système décimal au système hexadécimal s'accompagne d'une diminution du nombre des symboles employés. Par la combinaison de ces phénomènes, on peut s'attendre à ce que la conversion d'un nombre binaire en un nombre hexadécimal se traduise par une forte réduction du nombre de caractères. Nous constatons que la diminution s'opère par un facteur 4: X bits demanderont $\frac{X}{4}$ caractères hexadécimaux (chiffres).

Le système décimal nécessite 10 symboles de base (les chiffres de 0 à 9), le système binaire en demande deux (Ø et 1). Pour tout système de base inférieure à la base dix du système décimal, on dispose de caractères fondamentaux en nombre suffisant; il suffit de retenir les chiffres nécessaires, à partir de 0. En ce qui concerne le système hexadécimal, il n'est pas possible de prendre les nombres de 0 à 15, parce que chaque chiffre doit être représenté par un caractère, ou symbole, unique. C'est la raison pour laquelle on a choisi les chiffres de Ø à 9 complétés par les majuscules A, B, C, D, E, et F. L'équivalence des chiffres et des lettres est la suivante:

0 = 0 = ØØØØ	4 = 4 = Ø1ØØ	8 = 8 = 1ØØØ	C = 12 = 11ØØ
1 = 1 = ØØØ1	5 = 5 = Ø1Ø1	9 = 9 = 1ØØ1	D = 13 = 11Ø1
2 = 2 = ØØ1Ø	6 = 6 = Ø11Ø	A = 10 = 1Ø1Ø	E = 14 = 111Ø
3 = 3 = ØØ11	7 = 7 = Ø111	B = 11 = 1Ø11	F = 15 = 1111

Comment s'opère donc la conversion de la notation binaire (zéros et uns) à la notation hexadécimale? De la manière la plus simple. Le nombre binaire est partagé, de la droite vers la gauche, en groupes de quatre bits. A l'extrémité gauche du nombre binaire, il se peut qu'il ne reste qu'un, deux ou trois bits; les bits manquants sont remplacés par des zéros en nombre suffisant pour que le groupe le plus à gauche comporte, lui aussi, quatre bits. Ensuite, chaque quartet est remplacé par le caractère hexadécimal correspondant; enfin, tous les caractères hexadécimaux sont regroupés. Par exemple:

$$\underbrace{(Ø)11Ø}_{6}\underbrace{1Ø1Ø}_{A}\underbrace{1Ø11}_{B}\underbrace{11ØØ}_{C} = 6ABC_{16}$$

L'indice 16 inscrit à la partie inférieure droite de l'ensemble 6ABC signifie qu'il s'agit d'une notation hexadécimale. Son utilisation ne s'impose que

lorsqu'il est fait référence à plusieurs systèmes numériques différents dans le cours de calculs. Quand il ne peut y avoir erreur d'interprétation, il est fréquent qu'on ne le fasse pas figurer.
Afin de prévenir tout malentendu, il est bon de noter que la notation hexadécimale s'applique aussi aux ensembles de bits (codes numériques) non considérés comme des nombres binaires. En outre, en la circonstance il ne s'agit pas tant d'une sorte de notation sténographique que d'une notation intégrale dans le système hexadécimal. En d'autres termes, le nombre 6ABC est l'équivalent de:
$6x16^3 + Ax16^2 + Bx16^1 + Cx16^0 = 6x16x16x16 + 10x16x16 + 11x16 + 12 = 24\,576 + 2\,560 + 176 + 12 = 27324_{10}$ (en numération décimale).
Il est possible de contrôler ce résultat en convertissant le nombre binaire en nombre décimal.
Le fait que la répartition en groupes de quatre bits soit particulièrement efficace vient de ce que la *base* du nouveau système (16) est une puissance de la base du système de numération initial (2). Pour la conversion du binaire à l'octal (système à 8 chiffres), il faut constituer des groupes de trois bits; pour la conversion du binaire au système de base 32, les groupes seront de 5 bits.
La figure 1 donne un aperçu des trois systèmes numériques qui ont le plus d'importance pour notre exposé. La figure 2 illustre l'une des méthodes utilisables à la conversion d'un nombre décimal en un nombre binaire équivalent. Le nombre est divisé successivement par deux. Lorsqu'il s'agit d'un nombre impair, le reste est égal à 1; la division d'un nombre pair par 2 provoque l'apparition d'un reste égal à Ø. Le code obtenu en colonne est ensuite réécrit en ligne. Il est possible de convertir directement un nombre décimal en un nombre hexadécimal, mais il est souvent plus commode de procéder d'abord à la conversion en binaire, puis de continuer par le fractionnement du nombre binaire obtenu en groupes de quatre bits, et d'inscrire l'équivalent hexadécimal de chacun de ces groupes.

Calculs binaires

Pour l'essentiel, il existe une grande analogie entre les calculs décimaux que nous connaissons bien et les calculs binaires. Les quatre opérations arithmétiques que sont l'addition, la soustraction, la multiplication et la division vont nous le montrer.

Addition binaire

Dans l'addition décimale, dès que le total de l'addition des chiffres d'une colonne atteint 10, il y a lieu de faire une retenue, laquelle est immédiatement additionnée au total des chiffres de la colonne directement à gauche de la précédente:

```
  129   premier nombre
  243   second nombre
+   1   retenue
 ----
  372   résultat = somme
```

binaire	décimal	hexa-décimal	quartet
Ø	Ø	Ø	ØØØØ
1	1	1	ØØØ1
1Ø	2	2	ØØ1Ø
11	3	3	ØØ11
1ØØ	4	4	Ø1ØØ
1Ø1	5	5	Ø1Ø1
11Ø	6	6	Ø11Ø
111	7	7	Ø111
1ØØØ	8	8	1ØØØ
1ØØ1	9	9	1ØØ1
1Ø1Ø	1Ø	A	1Ø1Ø
1Ø11	11	B	1Ø11
11ØØ	12	C	11ØØ
1101	13	D	1101
111Ø	14	E	111Ø
1111	15	F	1111
1ØØØØ	15	1Ø	
1ØØØ1	17	11	
.	.	.	
.	.	.	
.	.	.	
.	.	.	

Figure 1. Vue d'ensemble des systèmes binaire, décimal et hexadécimal. Dans le système binaire, tous les zéros situés à la gauche du 1 le plus à gauche d'un nombre sont éliminés. C'est d'ailleurs la règle dans le système décimal. Nous écrivons "9" et non pas "09".

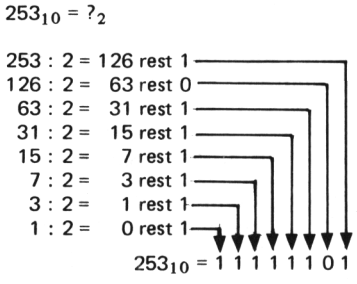


Figure 2. La conversion d'un nombre décimal en un nombre binaire équivalent s'effectue tout simplement par une suite de divisions par 2. Lorsque, dans la série de divisions, l'un des diviseurs est un nombre impair, le reste est égal à 1; quand les diviseurs sont pairs, le reste est égal à Ø. Le nombre binaire est constitué de la série des restes notés de la droite vers la gauche, dans l'ordre de leur apparition.

Dans l'addition binaire, la retenue apparaît dès que le total des chiffres d'une colonne est égal à 2.

```
  1Ø11Ø1   premier nombre
   1Ø1Ø1   second nombre
+ 1111 1   retenue
---------
 1ØØØØ1Ø   résultat = somme
```

Dans le langage des informaticiens, il est d'usage d'employer le mot américain *carry* pour désigner une retenue. En fait, pour l'exécution d'une addition binaire, il suffit de se souvenir des règles suivantes:

```
1 + Ø = 1
Ø + 1 = 1
1 + 1 = Ø  avec carry, donc Ø1 + Ø1 = 1Ø
Ø + Ø = Ø
```

Ces règles peuvent être vérifiées en les appliquant à la figure 1 dans laquelle tout nombre binaire résulte de l'addition de 1 au nombre précédent.

Soustraction binaire

Dans la soustraction décimale, dès que la différence entre le chiffre du plus grand nombre et le chiffre de la même colonne du nombre soustrait est négative, un "1" est ajouté au chiffre du plus petit nombre placé dans la colonne immédiatement à gauche. En fait, il s'agit d'un report négatif correspondant à "l'emprunt" d'un 1.

```
  1984   (plus grand nombre)
   199   (nombre à soustraire)
-   11   emprunt
------
  1785   résultat = différence
```

La soustraction binaire s'effectue comme suit:

```
  11ØØØØØ1   plus grand nombre
   1111111Ø  nombre à soustraire
- 111111     emprunt
-----------
 Ø1ØØØØ11    résultat = différence
```

Pour désigner un "emprunt", les informaticiens utilisent le mot américain *borrow*. Quatre règles sont à observer:

```
Ø - Ø = Ø
1 - 1 = Ø
1 - Ø = 1
Ø - 1 = 1  avec borrow, donc 1Ø - Ø1 = Ø1
```

Par conséquent, une retenue (carry) provoque, dans l'addition, l'adjonction d'un 1 aux bits, à l'emplacement situé à gauche; un emprunt (borrow) entraîne, dans la soustraction, le retranchement d'un 1 supplémentaire de la différence de deux bits, à l'emplacement situé à gauche. Les quatre règles de la soustraction sont également vérifiables en les appliquant à

la figure 1. Chaque nombre précédent résulte de la soustraction de 1 du nombre suivant.

Multiplication binaire

Procédons à la multiplication de deux nombres décimaux:

```
   233
 x 147
 -----
  1631
  932
+ 233
 -----
 34251
```

L'analyse de l'opération montre que l'on calcule d'abord le résultat de la multiplication de 7 par 233; puis on calcule le résultat de la multiplication de 4 par 233, mais que l'on décale en totalité d'une position vers la gauche, de sorte qu'il s'agisse en réalité du résultat de 40 x 233. Enfin, le résultat de 1 x 233 est décalé en totalité de deux positions vers la gauche, ce qui équivaut à 100 x 233. On effectue alors l'addition de 1631+9320+23300, ce qui donne le résultat, 34251. Il en va de même pour la multiplication des nombres binaires. C'est même plus simple, parce qu'il n'y a que des multiplications de 1 ou de Ø:

```
    1Ø11    multiplicande (11₁₀)
x   1Ø1Ø    multiplicateur (10₁₀)
 ---------
    ØØØØ
   1Ø11
  ØØØØ
 1Ø11
+  1        retenue
 ---------  résultat (110₁₀)
 11Ø111Ø
```

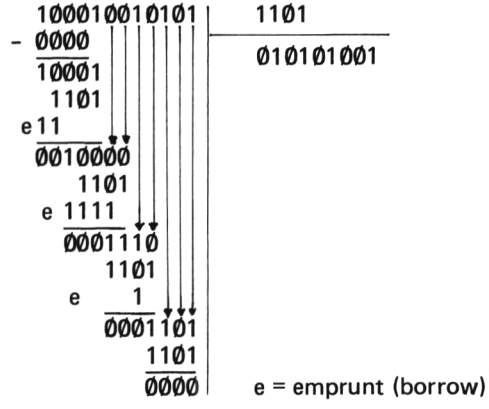
Lorsqu'il s'agit de multiplier, d'additionner ou de soustraire des nombres hexadécimaux, il suffit de commencer par les écrire en binaire, en convertissant les caractères en leur groupe de quatre bits correspondant. Les résultats des opérations effectuées en binaire sont ensuite reconvertis en hexadécimal.

Division binaire

Voyons d'abord la division décimale de 2091 par 17

```
  2091 | 17
- 17   |---
 ---   | 123
  39
- 34
 ---
   51
- 51
 ---
    0
```

Pour la division binaire, le processus est identique. A l'inverse de la multiplication, la division comporte une suite de soustractions et de décalages à droite. A chaque fois, il y a comparaison entre le reste et le diviseur. Si le reste est plus grand que le diviseur, on note 1 au quotient; un nouveau reste apparaît, résultant de la soustraction du diviseur du reste précédent. On ajoute le bit suivant du dividende à la droite du nouveau reste.
Par contre, si le reste est plus petit que le diviseur, c'est un Ø qui est inscrit au quotient et l'on abaisse le bit suivant du dividende, à la droite du reste.
Examinons un exemple:
Dividende: 1ØØØ1Ø, soit 2197_{10}
Diviseur: 11Ø1, soit 13_{10}
Quotient: 1Ø1Ø1ØØ1, soit 169_{10}



Nombres négatifs

Jusqu'à présent, nous avons omis délibérément de spécifier que les opérations ci-dessus portaient sur des nombres binaires positifs. Cependant, de même qu'il existe des nombres décimaux négatifs, il y a des nombres binaires négatifs. Différentes méthodes autorisent la représentation des nombres négatifs. Nous nous limiterons à l'étude de la méthode du *complément à deux.*
Le Junior Computer, et plus précisément le microprocesseur 6502, permet le traitement de mots d'un format de 1 octet, soit 8 bits. Sans prendre de dispositions particulières, il est donc possible de représenter tous les nombres binaires allant de ØØØØØØØØ à 11111111 (ou encore, les nombres hexadécimaux ØØ . . . FF, les nombres décimaux 0 . . . 255). Pour les nombres supérieurs, il faut accoler plusieurs octets.
La figure 3 représente les nombres binaires de ØØØØØØØØ à 11111111 présentés sur un arbre de nombres. On peut l'assimiler à une jauge graduée partagée en 256 unités de longueur. La distance entre deux nombres équivaut à l'addition de deux distances. La distance totale peut donc éventuellement dépasser les 256 unités de longueur. Cependant, la somme

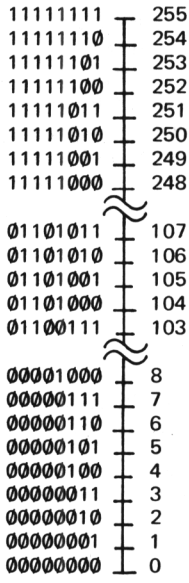


Figure 3. Arbre des nombres binaires à 8 bits, positifs.

binaire ne peut être supérieure à 11111111, parce qu'on ne dispose pas d'un neuvième bit. Il n'empêche que nous verrons au chapitre 3 que la retenue relative au neuvième bit, lequel n'est pas présent, sera quand même enregistrée grâce à l'indicateur (flag) Carry, ce qui nous permettra de constituer un second octet 00000001 prenant place à la gauche du premier. Imaginons que nous ne fassions usage que de 7 bits d'un octet. Cela signifie que le domaine numérique utilisable sera limité aux nombres allant de 00000000 à 01111111, autrement dit, de zéro à 127 inclus. Ce qui correspond à 128 des 256 possibilités d'un nombre à 8 bits.
Tentons maintenant de soustraire 1 de 0

```
            00000000
            00000001
emprunt     11111111
          - --------
            11111111   ou, zéro moins un = moins un, ou -1
            00000001   à nouveau, soustraction de 1
emprunt
          - --------
            11111110                              ou -2
            00000001   à nouveau, soustraction de 1
emprunt           1
          - --------
            11111101                              ou -3
```

C'est ainsi que les nombres entiers négatifs peuvent être représentés directement, ainsi que le montre la figure 4, qui reproduit quelques uns des

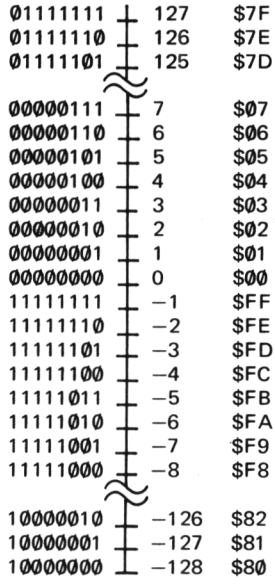


Figure 4. Arbre des nombres binaires positifs et négatifs à 8 bits; ces derniers sont obtenus par la méthode du complément à deux. A droite, sont inscrits les nombres hexadécimaux correspondants.

nombres entiers de +127 à - 128. Les nombres négatifs sont reproduits à l'aide de la méthode du complément à deux. Il n'est pas dans notre intention de vous infliger la description de la totalité des théories des nombres à la base de cette méthode. Il nous paraît bien préférable de vous indiquer uniquement comment procéder très simplement pour trouver rapidement la représentation binaire d'un nombre négatif.

1. D'abord nous déterminons la valeur absolue du nombre - a en notation binaire.
2. Ensuite, nous transformons tous les Ø de ce nombre binaire en 1, et tous les 1 en Ø.
3. A cette nouvelle écriture du nombre, nous ajoutons 1, ce qui nous donne finalement, la représentation binaire du nombre - a, en complément à deux.

Par exemple, quelle est la représentation binaire du nombre entier négatif - 3?

Plus trois (+3) s'écrit ØØØØØØ11 en binaire. Procédons à l'interversion des 1 et des Ø: 11111ØØ. A ce nombre, ajoutons ØØØØØØØ1, ce qui nous donne 11111Ø1, soit - 3 en notation binaire (voir également la figure 4).

C'est, par conséquent, au prix d'un bit, ou encore au prix du sacrifice du nombre maximal des différentes possibilités de représentation des entiers *positifs*, pour une longueur de mot déterminée, que l'on arrive à représenter également les entiers négatifs. Les nombres négatifs commencent toujours par un 1 (le bit le plus à gauche); les nombres positifs

binaire	décimal	hexa-décimal
11111111	– 1	FF
1111111Ø	– 2	FE
11111101	– 3	FD
11111100	– 4	FC
11111011	– 5	FB
1111101Ø	– 6	FA
11111001	– 7	F9
11111000	– 8	F8
11110111	– 9	F7
1111Ø11Ø	–1Ø	F6
11110101	–11	F5
11110100	–12	F4
11110011	–13	F3
1111ØØ1Ø	–14	F2
11110001	–15	F1
1111ØØØØ	–16	FØ
11101111	–17	EF
1110111Ø	–18	EE
11101101	–19	ED
11101100	–2Ø	EC
1110101Ø	–21	EB
11101010	–22	EA
11101001	–23	E9
11101000	–24	E8
11100111	–25	E7
1110011Ø	–26	E6
11100101	–27	E5
11100100	–28	E4
11100011	–29	E3
1110001Ø	–3Ø	E2
11100001	–31	E1
11100000	–32	EØ
11011111	–33	DF
1101111Ø	–34	DE
11011101	–35	DD
11011100	–36	DC
1101101Ø	–37	DB
11011010	–38	DA
11011001	–39	D9
11011000	–4Ø	D8
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.

Figure 5. Extrait de la table des nombres binaires négatifs avec les nombres décimaux et hexadécimaux correspondants.

ayant la *même* longueur de mot commencent toujours par un Ø qu'il est tout à fait possible de supprimer. La figure 5 présente quelques nombres binaires négatifs ainsi que leurs équivalents hexadécimaux.

Code DCB

En l'occurrence, *il ne s'agit pas* d'un système numérique, mais d'un code dans le sens le plus classique du terme. Chaque chiffre d'un nombre décimal est traduit par un groupe de 4 bits. Chaque groupe de quatre bits est la représentation binaire de chaque chiffre. Par conséquent, on obtient:

0 = ØØØØ
1 = ØØØ1
2 = ØØ1Ø
3 = ØØ11
4 = Ø1ØØ
5 = Ø1Ø1
6 = Ø11Ø
7 = Ø111
8 = 1ØØØ
9 = 1ØØ1

Comme vous le constatez, le codage est le même que pour les dix premiers chiffres du système hexadécimal. La représentation DCB d'un nombre résulte de l'écriture côte à côte, et dans le même ordre que celui du nombre initial, des groupes de quatre bits. Par exemple, la représentation DCB de 1981 est:

ØØØ11ØØ11ØØØØØØ1

1 9 8 1

Cela ne ressemble pas le moins du monde au code binaire de 1981, qui, ainsi que nous l'avons vu, est:

1111Ø111Ø1

L'avantage du code DCB est qu'il permet de distinguer la valeur décimale du nombre beaucoup plus facilement; on obtient des résultats absolument exacts. L'inconvénient de ce mode de calcul est qu'il est plus lent que le binaire classique. Mais il n'est pas impossible de l'utiliser, témoin le fait que le microprocesseur 6502 du Junior Computer le permet.

L'ordinogramme

A première vue, il se peut que le paragraphe qui débute semble ne pas s'inscrire dans le cadre du chapitre 2 où il a sourtout été question de divers systèmes de numération et de la représentation binaire des nombres décimaux. Il n'en est pas moins vrai que, avant d'aborder le chapitre 3, il est utile de disposer de connaissances de base nécessaires à la programmation.

Quelle est la justification de l'emploi d'un ordinateur, et tout spécialement du Junior Computer, sinon la résolution d'un problème. Il tombe sous le sens que le mot "problème" doit être interprété dans sa signification la plus large. C'est à l'utilisateur qu'importe la résolution du problème. Le

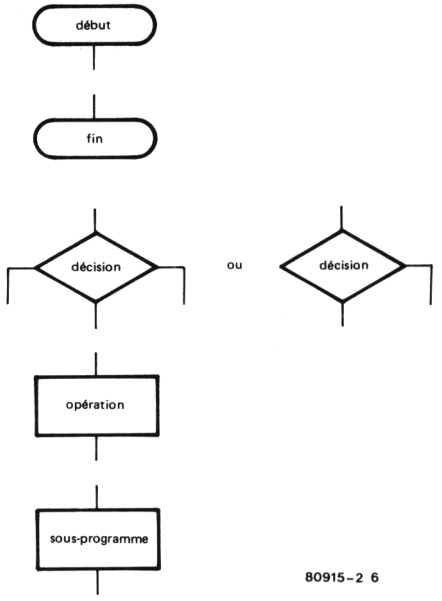


Figure 6. Figures géométriques symboliques les plus fréquemment utilisées dans la composition des ordinogrammes. Il en existe d'autres, et il faut savoir que, parfois, divers symboles s'appliquent à une fonction identique. Notez bien que le losange à trois sorties ne s'emploie que dans un ordinogramme très général.

Junior Computer, quant à lui, fournira la solution en fonction des indications que lui aura données l'utilisateur sous la forme d'instructions. La traduction des directives en instructions concrètes constitue la matière de la programmation. Ce sera le sujet du chapitre 3. Le présent paragraphe traite de l'auxiliaire précieux qu'est *l'ordinogramme* (que l'on désigne aussi sous le nom d'organigramme ou flow chart) pour la détermination des directives, et, par conséquent, pour l'obtention de la solution.

Un ordinogramme est la représentation graphique d'un algorithme établi à partir du processus retenu pour la résolution d'un problème, lequel, très souvent, est justifiable de plusieurs solutions.

A tout problème il existe un commencement et une fin; entre ces deux extrémités, il se déroule un certain nombre d'actions sous forme de décisions et d'opérations. La figure 6 montre les symboles des blocs fonctionnels composant un ordinogramme. Les décisions sont représentées par un losange; selon les conditions, il est possible d'emprunter deux ou trois voies différentes. Le rectangle, dans lequel s'inscrivent deux lignes parallèles aux largeurs, indique qu'il s'agit d'un sous-programme ou programme partiel. Un sous-programme fait lui-même l'objet d'un ordinogramme. Il y a lieu, en tout cas, d'esquisser sommairement, au préalable, les grandes lignes de la résolution du problème, en composant un ordinogramme. Cela signifie que le problème est fractionné en un certain nombre d'opérations et de décisions essentielles. Ensuite, on remplit

les rectangles et les losanges, ce qui permet d'établir un ordinogramme plus détaillé. Au fur et à mesure que celui-ci s'affine, les instructions destinées au Junior Computer se multiplient et se précisent et l'on en arrive au point où chaque rectangle, chaque losange comporte une directive pour le Junior-Computer.

Exemple d'ordinogramme

A titre d'illustration de la notion d'ordinogramme, celui que nous présentons en figure 7, sous une forme plus ou moins élaborée, concerne la solution d'un problème que vous aurez peut-être résolu avec la lecture de

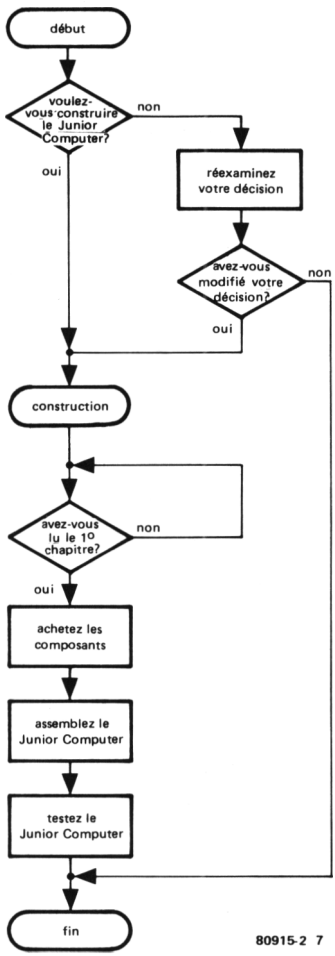


Figure 7. Ordinogramme général de la construction du Junior Computer. Cet ordinogramme de base est le point de départ d'un ordinogramme plus élaboré.

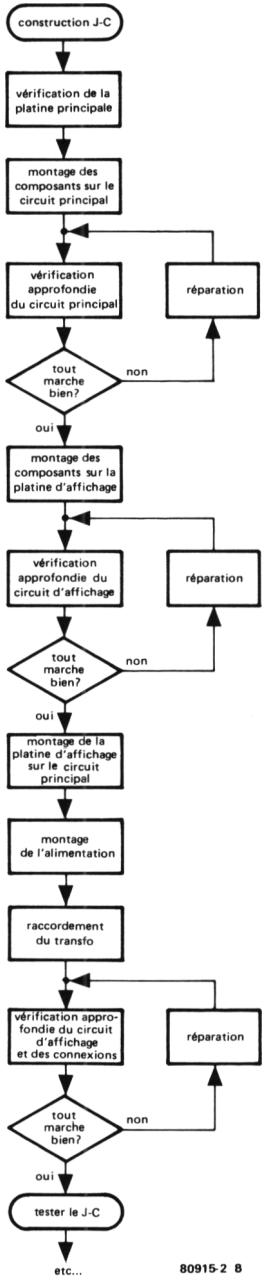


Figure 8. Ordinogramme élaboré de l'opération "Construction du Junior Computer" présenté dans l'ordinogramme général de la figure 7.

cet ouvrage: la construction du Junior Computer, telle qu'elle est décrite dans le premier chapitre de ce livre.
Vous constatez qu'après que la décision de construire le Junior Computer ait été prise, et que les instructions de montage aient été lues, les opérations "partager en sous-programmes", "construire le Junior Computer" et "tester le Junior Computer" sont exécutées et l'on approche de "fin". Il existe une autre voie pour parvenir à cette extrémité, si, par exemple, on décide de ne pas entreprendre la construction.
La liaison entre une sortie et une entrée d'un losange est un exemple de ce qu'est une "boucle d'attente". C'est seulement après que le "oui" ait été validé qu'il est possible d'accéder à l'opération suivante du programme.
La figure 7 est l'illustration d'un ordinogramme très général. Celui de la figure 8 est beaucoup plus élaboré et concerne l'opération "construction

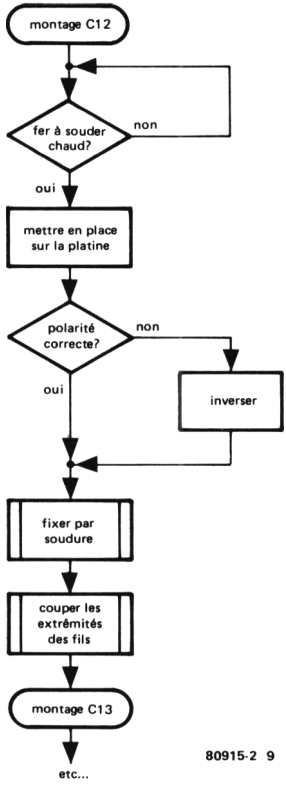


Figure 9. Extrait de l'ordinogramme de l'opération "Montage des composants sur le circuit principal" de la figure 8. Il s'agit de l'ordinogramme de l'opération "Montage de C12". Trois manipulations prévues dans cet ordinogramme sont réunies en un sous-programme, afin que l'ensemble puisse être utilisé à nouveau dans des opérations identiques se reproduisant souvent dans l'exécution de l'ordinogramme très détaillé, relatif au montage des composants sur la platine du circuit imprimé.

du Junior Computer", très schématisée dans l'ordinogramme de la figure 7. Dans la figure 8, vous retrouvez esquissé à grands traits ce qui est décrit de manière plus détaillée dans le premier chapitre.
Franchissons encore une étape. Le montage du condensateur électrolytique C12 fait partie de l'opération plus générale inscrite dans le rectangle "Composants du circuit principal". L'ordinogramme de la figure 9 détaille le montage du composant C12. Trois manipulations se rapportant à l'exécution de cette opération sont regroupées dans un sous-programme. Cela tient au fait que ces manipulations seront aussi utilisées dans un grand nombre d'autres opérations effectuées pour le montage d'autres composants sur les circuits imprimés. Il apparaît que les sous-programmes présentent parfois des avantages importants comparativement aux programmes classiques. Le chapitre 3 nous en donnera l'explication.

Exercices

Reconnaissons que le chapitre qui s'achève a été parfois très absorbant; mais il était indispensable que nous nous adaptions au traitement des 1 et des Ø. C'est d'ailleurs la raison qui justifie la présence des exercices que nous vous suggérons instamment de faire, afin que vous soyez parfaitement familiarisé avec le type de "pensée" du Junior Computer.

1) *Quelle est la notation hexadécimale de ØØ1Ø1111?*
2) *„ „ „ „ 11111?*
3) *„ „ „ „ 1Ø1ØØØ111?*
4) *„ „ „ „ 11Ø1Ø1Ø1Ø?*
5) *„ „ „ „ ØØ1Ø?*
6) *„ „ „ „ ØØ1Ø11?*
7) *Notation binaire du nombre hexadécimal 132?*
8) *„ „ „ „ AØ14?*
9) *„ „ „ „ Ø356?*
10) *„ „ „ „ A561?*
11) *„ „ „ „ ABBA?*
12) *Résultat de Ø1ØØ1111+11ØØØ111?*
13) *„ „ 111ØØ11+11111111?*
14) *„ „ 11111111+1?*
15) *„ „ 1111ØØØØ+1111?*
16) *„ „ 1Ø1Ø1Ø1Ø+1Ø1Ø1Ø1?*
17) *„ „ Ø1111Ø1ØØ-11Ø1?*
18) *„ „ 11111ØØØ-1111?*
19) *„ „ 1Ø111ØØØ-1ØØØØØØ1?*
20) *„ „ 1Ø1Ø1111-1Ø1Ø1111?*
21) *„ „ 1ØØ-11111Ø11?*
22) *„ „ 1111ØØØ1xØ1111?*
23) *„ „ 1Ø1x11111111?*
24) *„ „ 1Ø1Øx1Ø1Ø?*
25) *„ „ 11x11111111?*
26) *Produit de la multiplication des hexadécimaux 4 et ABBA?*
27) *Résultat de 1ØØØØØØØ101 : 111?*
28) *„ „ 11Ø1ØØØØØØØ:11Ø1?*
29) *„ „ 1ØØ11Ø1111Ø11ØØ1Ø:1ØØ111Ø?*
30) *„ „ $B9AØ:$ØB?*

Réponses aux exercices

1) *2F*
2) *1F*
3) *147*
4) *1AA*
5) *2*
6) *ØB*
7) *ØØØ1ØØ11ØØ1Ø*
8) *1Ø1ØØØØØØØ1Ø1ØØ*
9) *ØØØØØØ11Ø1Ø1Ø11Ø*
10) *1Ø1ØØ1Ø1Ø11ØØØØ1*
11) *1Ø1Ø1Ø111Ø111Ø1Ø*
12) *1ØØØ1Ø11Ø*
13) *1Ø111ØØ1Ø*
14) *1ØØØØØØØØ*
15) *11111111*
16) *11111111*
17) *11ØØ111*
18) *1110ØØØ1*
19) *ØØ11Ø111*
20) *Ø*
21) *1ØØØØ1ØØ1 (- 247 en 9 bits, par la méthode du complément à deux)*
22) *111ØØØØ11111*
23) *1ØØ11111Ø11*
24) *11ØØ1ØØ*
25) *1Ø11111Ø1*
26) *1Ø1Ø1Ø111Ø111Ø1ØØØ = $2AEE8*
27) *1ØØ1ØØ11*
28) *1ØØØØØØØØ*
29) *1111111111*
30) *1ØØØØ1110ØØØØ = $1ØEØ*

3

Programmation...

ou du bon usage d'un langage!

La construction du Junior Computer étant achevée et le chapitre 2 nous ayant permis d'amasser une quantité d'informations fondamentales, nous en venons maintenant à leur mise en oeuvre. Le chapitre qui débute devrait permettre de répondre très complètement aux deux questions que nous nous posons: qu'en faire, et, comment le faire? Grâce à quoi nous découvrirons comment tirer le meilleur parti de l'"interface" entre l'utilisateur et son Junior Computer.

Avant d'entreprendre la programmation du Junior Computer, raccordons-le au réseau grâce à la fiche et à la prise de courant, et fermons l'interrupteur secteur. Le moment est venu de nous concentrer sur notre tâche, tout en sachant très bien, qu'en peu de temps, et pourvu que nous persévérions, la programmation ne sera plus qu'un jeu, sérieux, certes, mais dont nous exploiterons aisément toutes les règles.

Reconnaissance du terrain

Réexaminons d'abord rapidement les 23 touches du clavier, les deux commutateurs à bascule et les six afficheurs à sept segments; la figure 1a nous rappelle la présentation de ce "tableau de bord". Puisque notre Junior Computer est raccordé au secteur et que le commutateur général est fermé, appuyons sur la touche RST. Les afficheurs s'illuminent et des caractères hexadécimaux quelconques apparaissent. Cette pression sur la touche RST a pour effet d'initialiser le programme moniteur mémorisé dans l'EPROM 2708. Dès cet instant, l'ordinateur interroge chacune des touches pour savoir si l'une d'entre elles a été pressée. Si c'est le cas pour l'une des touches Ø . . . F (caractères hexadécimaux), le caractère correspondant est affiché.
Les quatre afficheurs les plus à gauche indiquent en notation hexadécimale l'adresse d'une donnée. En principe, toutes les adresses allant de ØØØØ à

FFFF sont utilisables. Les deux afficheurs les plus à droite donnent le contenu de l'emplacement mémoire déterminé par l'adresse.
Imaginons précisément que nous voulions introduire des données (en notation hexadécimale) en des emplacements déterminés de la mémoire. Les nombres hexadécimaux seront, par exemple, 18, A9, Ø3, 69, Ø4, 8D, ØØ, Ø3, 4C, ØØ, Ø3. Le premier nombre, 18, sera introduit à l'adresse Ø2ØØ et les autres le seront aux adresses suivantes. Voici le tableau des opérations à effectuer:

	Touches pressées		Affichage		Mémoire		
			adresse	donnée			
RST			xxxx	xx			
AD			xxxx	xx	Ø1FD	X X	
Ø	2	Ø	Ø	Ø2ØØ	xx	Ø1FD	X X	
DA				Ø2ØØ	xx	Ø1FF	X X	
		1	8	Ø2ØØ	18	Ø2ØØ	X X	
+		A	9	Ø2Ø1	A9	Ø2ØØ	1 8	CLC
+		Ø	3	Ø2Ø2	Ø3	Ø2Ø1	A 9	LDA
+		6	9	Ø2Ø3	69	Ø2Ø2	Ø 3	
+		Ø	7	Ø2Ø4	Ø7	Ø2Ø3	6 9	ADC
+		8	D	Ø2Ø5	8D	Ø2Ø4	Ø 7	
+		Ø	Ø	Ø2Ø6	ØØ	Ø2Ø5	8 D	STA
+		Ø	3	Ø2Ø7	Ø3	Ø2Ø6	Ø Ø	
+		4	C	Ø2Ø8	4C	Ø2Ø7	Ø 3	
+		Ø	Ø	Ø2Ø9	ØØ	Ø2Ø8	4 C	JMP
+		Ø	3	Ø2ØA	Ø3	Ø2Ø9	Ø Ø	
AD						Ø2ØA	Ø 3	
1	A	7	A	1A7A	xx	Ø2ØB	X X	
DA				1A7A	xx	Ø2ØC	X X	
		Ø	Ø	1A7A	ØØ	1A79	X X	
+		1	C	1A7B	1C	1A7A	Ø Ø	
						1A7B	1 C	
						1A7C	X X	

N.B. Les caractères entourés d'un rectangle correspondent aux touches à presser, en partant de la gauche vers la droite.

La pression sur la touche RST a donc fait apparaître une série de caractères hexadécimaux quelconques. Il est bien évident que les afficheurs ne font pas apparaître des "X", mais des caractères allant de Ø à F, et qui, dans l'exemple présent, n'ont absolument aucune importance. Nous avons imaginé que le programme moniteur a été initialisé et que l'adresse de l'emplacement mémoire visualisé par les afficheurs était Ø1FD, le contenu de la mémoire, pour cette adresse étant quelconque (XX).

- Une pression sur la touche AD indique à l'ordinateur que nous sélectionnons le mode "adresse" et que les caractères hexadécimaux, allant de Ø à F, que nous allons taper ensuite, formeront une adresse à laquelle

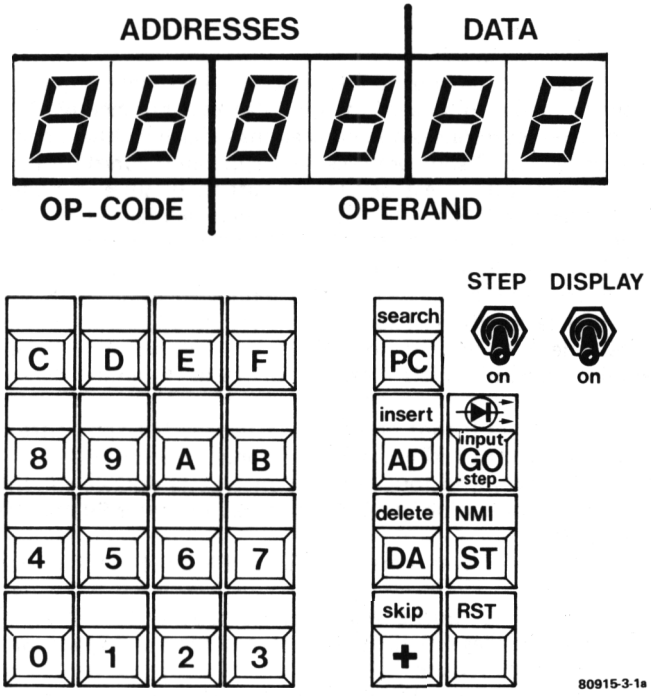


Figure 1a. "Tableau de bord" du Junior Computer. Il regroupe seize touches hexadécimales, sept touches et deux commutateurs de fonctions. Le texte en petits caractères surmontant chaque touche de fonction indique la fonction exercée, lorsqu'on veut rédiger ("editing") un programme. Un chapitre prochain abordera ce point en détail. Les expressions "op-code" (code opération) et "operand" (opérande) s'y rapportent également. En utilisation normale, les afficheurs Di1 à Di4 sont réservés à l'affichage d'une adresse en un instant donné; les afficheurs Di5 et Di6 sont destinés aux données (le format est celui d'un octet). Les adresses et les données sont affichées en code hexadécimal.

une donnée devra être stockée ultérieurement. Dans le cas présent, cette adresse est Ø2ØØ.

- Nous pressons, l'une après l'autre et dans l'ordre, les touches Ø2ØØ. Le premier Ø tapé apparaît sur le quatrième afficheur en partant de la gauche, puis, lorsque nous pressons la touche 2, ce Ø est décalé pour venir sur le troisième afficheur en partant de la gauche, tandis que le chiffre 2 est visualisé par le quatrième afficheur à partir de la gauche. Quand nous tapons le troisième chiffre (Ø), le Ø et le 2 sont décalés simultanément sur le second et le troisième afficheurs à partir de la gauche; en même temps, le troisième chiffre (Ø) est visualisé par le quatrième afficheur en partant de la gauche. Finalement, le quatrième chiffre (Ø) est tapé, les trois chiffres déjà visualisés, Ø2Ø, sont tous décalés vers la gauche et la totalité de l'adresse, Ø2ØØ, s'inscrit dans le cadre de l'affichage d'adresse. L'affichage

des données fait apparaître simultanément des données quelconques stockées à l'adresse Ø2ØØ.

- Une pression sur la touche DA indique à l'ordinateur que nous allons utiliser le mode "données" et, qu'à l'adresse affichée (Ø2ØØ), devront être inscrites les données que nous allons taper immédiatement après.
- Les touches 1 et 8 sont pressées; le 1 apparaît d'abord sur l'afficheur le plus à droite, puis il est décalé sur l'afficheur situé immédiatement à gauche, tandis que le 8 s'inscrit sur l'afficheur le plus à droite. Désormais, l'affichage fait apparaître Ø2ØØ 18 (l'adresse et la donnée écrite à cette adresse), car, dans le même temps, le contenu antérieur de l'adresse Ø2ØØ (XX) a été effacé pour que soit inscrite la nouvelle donnée, 18.
- On presse la touche +; l'adresse affichée, Ø2ØØ, est incrémentée (augmentée) de 1 et devient donc Ø2Ø1 (c'est l'adresse de l'emplacement mémoire suivant), mais l'ordinateur reste toujours dans le mode "données".
- Les touches A et 9 sont pressées. L'affichage fait apparaître Ø2Ø1 A9, ce qui indique qu'à l'adresse Ø2Ø1, la donnée A9 est inscrite.
- Ensuite, on presse la touche + et l'on tape la donnée suivante, Ø3; et l'on répète l'opération pour les données restantes 69, Ø4, 8D, ØØ, Ø3, 4C, ØØ et Ø3.

Supposons qu'après cela nous décidions d'inscrire les données ØØ, 1C etc. . . .

- Nous pressons la touche AD, puis les touches 1, A, 7, A, ensuite la touche DA, puis les touches ØØ, après quoi nous appuyons sur la touche + et nous tapons 1C, etc. . . .

Remarquons que lorsque nous pressons la touche AD, l'ordinateur reste au mode "adresse", et que, lorsque nous enfonçons la touche DA, il se maintient au mode "donnée".

Peut-être n'est-il pas inutile de récapituler ce que cette expérience nous a appris:

— les touches Ø à F servent à inscrire les adresses, et les données à emmagasiner à ces adresses.
— la touche RST sert à initialiser le programme moniteur et provoque l'illumination de l'affichage.
— la touche AD sélectionne le mode adresse permettant l'introduction d'une ou plusieurs adresses.
— la touche DA porte l'ordinateur au mode donnée; ensuite, l'appareil interprète toute pression sur l'un des 16 caractères hexadécimaux comme l'instruction d'introduire une donnée.
— une pression sur la touche + incrémente de 1 l'adresse affichée immédiatement avant que la touche ne soit enfoncée. Le mode (adresse ou donnée) ne subit aucune modification.

Faisons une courte vérification: pressons, dans l'ordre, les touches AD Ø 2 Ø Ø; l'affichage fait apparaître Ø2ØØ 18. C'est la preuve que la donnée 18 est toujours emmagasinée à l'adresse Ø2ØØ. Si nous appuyons ensuite plusieurs fois sur la touche +, chaque pression nous montrera que les autres données sont bien stockées aux emplacements mémoire choisis.

Ainsi, grâce à ces quelques manipulations, nous avons appris à nous servir du clavier et de l'affichage pour inscrire des données en mémoire, ainsi qu'à les y retrouver. Pour être complets, précisons maintenant que le

chargement que nous venons d'opérer ne concernait pas des données quelconques. Il s'agissait en fait d'un petit programme destiné à l'addition de deux nombres hexadécimaux. Sans vouloir entrer dès à présent dans les détails, voici quand même quelques indications.
Le nombre hexadécimal 18 est stocké à l'adresse Ø2ØØ; il est la formulation codée de l'instruction CLC (Clear Carry) soit, mise à zéro de la retenue. Puis, l'ordinateur avance à l'adresse Ø2Ø1 où est stocké le nombre A9, représentation codée de l'instruction LDA (LoaD Accumulator immediat) c'est-à-dire, chargement immédiat d'une nouvelle donnée dans l'accumulateur. Cette donnée est stockée à l'adresse Ø2Ø2, et Ø3 est donc chargé dans l'accumulateur. A l'adresse suivante, Ø2Ø3, est emmagasiné le nombre 69, code hexadécimal pour l'instruction ADC (ADdition with Carry), ce qui veut dire addition du contenu de l'emplacement-mémoire à l'adresse immédiatement suivante, avec celui de l'accumulateur. Le contenu de l'adresse Ø2Ø4 est Ø7 et celui de l'accumulateur Ø3. Son nouveau contenu devrait donc être ØA.
Mais, par curiosité, quel est le contenu de l'adresse Ø2Ø5? C'est 8D, expression codée de l'instruction STA (STore Accumulator in memory) soit, rangement du contenu de l'accumulateur en mémoire. En quel emplacement? L'adresse nécessaire se trouve consignée aux deux adresses Ø2Ø6 et Ø2Ø7, le dernier octet d'adresse ØØ étant suivi par le premier Ø3; l'écriture normale est donc Ø3ØØ. A l'adresse Ø2Ø8 est écrite la donnée 4C, code pour l'instruction JMP (JuMP) ou saut à une adresse qui se trouve à l'emplacement mémoire indiqué par le contenu des adresses Ø2Ø9 et Ø2ØA. Cet emplacement a donc pour adresse Ø3ØØ et son contenu est ØA. Ce nombre hexadécimal est l'expression codée de l'instruction ASL-A (Arithmetic Shift Left Accumulator), c'est-à-dire, décalage arithmétique à gauche du contenu de l'accumulateur, et livre le résultat de l'addition.
Pour le moment, il n'est pas encore question d'expliquer et de justifier les instructions du programme. Il s'agit plutôt de nous familiariser avec toutes sortes de données qui seront utilisées d'une manière ou d'une autre. Une donnée déterminée représente la forme codée d'une instruction précise. Celle-ci se rapporte à une donnée et, à cet égard, nous parlons d'un *opérande*: par exemple, quelle donnée doit être chargée ou mémorisée; ou, à quelle adresse?
Cet exemple de programme illustre la conception imaginée par von Neumann, il y a quelques décennies, et qui est celle du "stored program": des instructions et des opérandes codés de manière identique sont stockés dans une même mémoire. Le "cycle de von Neumann" se déroule comme suit: aller chercher dans la mémoire le code pour l'instruction (suivante) → décoder l'instruction (que doit-il être fait?)→ si nécessaire, aller chercher l'opérande (où l'instruction est-elle déposée?)→ exécuter l'instruction→ détecter l'adresse à laquelle se trouve l'instruction suivante→ aller chercher cette instruction→ et ainsi de suite.
Tout cela nous aide à mieux comprendre que le mot "programme" recouvre, en fait, une succession d'opérations dont l'ordre de déroulement doit être impérativement observé.

Programmation type du 6502

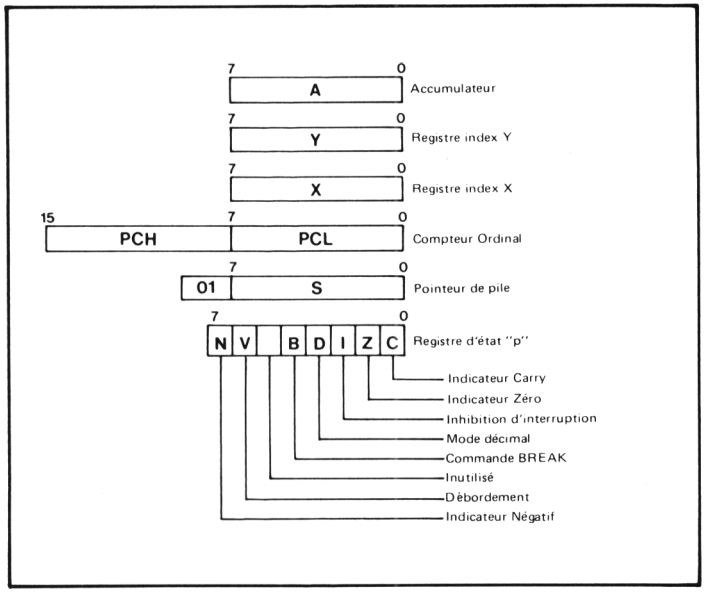
Avant de nous intéresser davantage aux possibilités offertes par le logiciel (software) du Junior Computer, ainsi que par le jeu d'instructions et les modalités d'adressage, il nous faut faire plus ample connaissance avec l'architecture interne du microprocesseur 6502, et avec ses registres. Il est évident que le contenu de ceux-ci sera déterminé par l'utilisateur, à l'aide du programme. Mais, avant que nous examinions comment cela se fera, il convient de savoir sur quoi s'appuieront les opérations. Reportons-nous à la figure 1b.

Le rectangle A représente l'*accumulateur* qui est un registre à 8 bits. Il sert à aller copier les données en mémoire, mais c'est également le point de départ du chargement de données en mémoire. Il est aussi utilisé comme intermédiaire assurant le transfert d'une donnée d'un emplacement mémoire à un autre.

Sept des huit bits que comporte le registre d'état P (status register) sont des indicateurs (flags). Ou'est-ce que cela? Ce sont des bits dont la valeur peut être "1" ou "Ø", en concordance avec l'état de la bascule dont ils dépendent. Les indicateurs peuvent être mis à l'état "1" ou remis à "Ø", correspondant aux vocables anglo-saxons "set" et "reset". Certaines instructions particulières permettent de placer les indicateurs dans un état ou l'autre, de manière directe et sans condition. D'autres instructions affectent le contenu des indicateurs de manière conditionnelle, c'est-à-dire que c'est le résultat de l'instruction (les données finales) qui influence l'état de certains indicateurs (voir à ce sujet la dernière colonne de l'appendice, pages . . . et . . .). Les indicateurs sont utilisés pour modifier le déroulement d'un programme lors de l'exécution de phases particulières de celui-ci. Un de ces indicateurs est le bit de retenue (ou "carry"). Il est placé à l'état haut (logique "1") lorsque le résultat de la sommation de deux nombres dans l'ALU exige un report du huitième bit (càd le bit situé à l'extrême gauche de l'octet) vers un neuvième bit. Un autre indicateur est, par exemple, N. Celui-ci indique l'apparition d'un résultat négatif. Quand à l'indicateur Z, son usage est d'indiquer que le résultat d'une opération est nul. Nous verrons plus tard la signification des indicateurs supplémentaires que comporte le registre P.

Le *compteur ordinal* (Program Counter), PC, est un registre de 16 bits contenant l'adresse de la prochaine instruction à exécuter (rappelons-nous le "cycle de von Neumann"). Il comporte deux registres dont l'un est le PCL (Program Counter Low) ou moitié basse qui renferme les 8 bits d'adresse les plus à droite (0 à 7), et l'autre est le PCH (Program Counter High) contenant les 8 bits les plus à gauche (8 à 15). Entre deux instructions, le PC spécifie l'adresse d'un emplacement mémoire où des données aussi bien que des opérandes pourront être copiées.

Les *registres index X et Y* contiennent également des données. Leur chargement est le résultat de l'exécution d'instructions. Il existe des techniques d'adressage déterminées (indexé et indirect) permettant de consigner certaines adresses. Le *pointeur de pile* (Stack Pointer) S spécifie les adresses nécessaires, lors du saut d'un programme à un sous-programme, au déroulement de certains opérations.



80915-3-1b

Figure 1b. Tableau du logiciel ("software") du Junior Computer, ou encore du microprocesseur 6502. Il donne un aperçu des registres internes dont le contenu est modifié par l'intermédiaire du logiciel, et, par conséquent, du programme. Le registre d'état, P, du microprocesseur, pourrait également s'appeler "registre des indicateurs".

Répertoire d'adressage du 6502

Treize modes d'adressage différents sont à la disposition de l'utilisateur du Junior Computer. C'est là l'un des grands avantages du 6502. Comparativement, le jeu d'instructions limité à 56 n'est pas si considérable. Mais, cette combinaison a pour mérite de faciliter très sensiblement la tâche du programmeur. Rares sont les autres fabricants ayant su atteindre ce résultat. Il est clair qu'un jeu d'instructions réduit est plus facilement mémorisé par l'utilisateur. Et cela d'autant mieux que toutes les instructions sont caractérisées par des mnémoniques (*mnemonics*, en américain) de trois lettres et par un code-opération (hexadécimal).

Adressage immédiat

Adressage direct

Les instructions utilisant l'adressage direct sont utilisées pour effectuer des opérations sur des données directement disponibles dans la mémoire de travail dès que l'instruction est connue. Elles sont constituées par deux

octets: le premier étant la description de l'instruction (code opération) et le second contenant l'opérande.
Il est caractérisé par le symbole # ("railroad crossing"). Par exemple:
LDA #7A signifie: charger la donnée 7A dans l'accumulateur;
LDX #3B: charger la donnée 3B dans le registre d'index X; au lieu du registre X, ce peut être Y.
Autre exemple:
ADC # A+M+C → A,
ADC # (octet) provoque l'opération suivante: A+M+C → A, ce qui veut dire: ajouter au contenu de l'accumulateur l'octet M, situé immédiatement après le code opération et ajouter de plus "1" ou "Ø", selon l'état de l'indicateur bit de retenue. A cela, ajouter le flag C (Carry) qui, avant une addition, doit toujours être remis à "Ø". Cela se fait grâce à l'instruction CLC (Clear Carry) signifiant la mise à zéro du bit de retenue.
Examinons d'un peu plus près la programmation d'une addition:

CLC	clear carry (C=Ø)
LDA#13	charger 13 dans l'accumulateur
ADC#Ø8	ajouter à cela Ø8
BRK	stopper dès que l'addition est terminée

Cette dernière instruction permet que le programme continue à se dérouler et soit arrêté lorsque le résultat recherché est atteint. Mais, dans le détail, comment allons-nous programmer le Junior Computer? D'abord, nous choisissons le code instruction (voir le tableau à la fin du livre). Nous obtenons: CLC = 18; LDA #A9; ADC #69; BRK = ØØ; l'adresse de début de programme sera ØlØØ. Ensuite, nous procédons comme nous l'avons déjà vu: mise sous tension, commutateur d'affichage (Display) sur ON, commutateur pas-à-pas (Step) sur OFF, puis:

Touches pressées				**Affichage** adresse	donnée	**Instruction**
RST	AD			xxxx	xx	
Ø	1	Ø	Ø	ØlØØ	xx	
DA		1	8	ØlØØ	18	CLC
+		A	9	Ø1Ø1	A9	LDA #
+		1	3	Ø1Ø2	13	
+		6	9	Ø1Ø3	69	ADC #
+		Ø	8	Ø1Ø4	Ø8	
+		Ø	Ø	Ø1Ø5	ØØ	BRK
AD				Ø1Ø5	ØØ	
1	A	7	E	1A7E	xx	
DA		0	0	1A7E	ØØ	
+		1	C	1A7F	1C	
AD						
Ø	1	Ø	Ø	ØlØØ	18	
GO				Ø1Ø7	xx	début de programme
AD				Ø1Ø7	xx	
Ø	Ø	F	3	ØØF3	1B	résultat

Il est vraisemblable que certains points vous paraissent quelque peu obscurs. Par exemple, pourquoi y a-t-il un saut brusque de l'adresse Ø1Ø5 à l'adresse 1A7E, et pourquoi le résultat apparaît-il à l'adresse ØØF3? En réalité, après l'exécution de l'instruction BRK (interruption software) à l'adresse Ø1Ø5, le programme lui-même est achevé. Ce qui suit est le déroulement d'une procédure indispensable, en rapport avec le moniteur du Junior Computer. Aux adresses 1A7E et 1A7F sont consignées les données 1CØØ qui constituent l'adresse de début d'un sous-programme du programme moniteur, qui est un programme de sauvegarde des données présentes dans tous les registres du microprocesseur (*Save Subroutine*). Le tableau ci-dessous montre la liste des emplacements mémoire où le contenu de chacun des registres sera déposé:
à l'adresse ØØEF, le contenu du registre PCL
à l'adresse ØØFØ, le contenu du registre PCH
à l'adresse ØØF1, le contenu du registre P
à l'adresse ØØF2, le contenu du registre S
à l'adresse ØØF3, le contenu du registre A
à l'adresse ØØF4, le contenu du registre Y
à l'adresse ØØF5, le contenu du registre X
C'est à l'adresse ØØF3 qu'est stocké le contenu de A, qui, dans notre exemple est le résultat, 1B, de l'addition de deux nombres. C'était d'ailleurs ce qu'indiquait la dernière ligne de programme. L'avant-dernière ligne concernait l'introduction de l'adresse de début du programme, suivie d'une pression sur la touche GO, grâce à quoi le programme est effectivement lancé.
Si nous réussissons à additionner, nous pouvons aussi faire des soustractions.
Dans ce cas, l'instruction à utiliser est SBC (Soustraction avec retenue), suivie de A-M-$\overline{C}$→ A. Ce qui veut dire: soustraire du contenu de l'accumulateur A le nombre hexadécimal faisant suite immédiatement, dans la mémoire M, à l'instruction de soustraction. Cela se fera sous la forme du complément à deux (voir le chapitre 2). Soustraire ensuite le contenu C du flag Carry. Le résultat exact d'une soustraction de deux nombres demande que le flag Carry soit mis à "1", par conséquent $\overline{C}$ = Ø et C = 1. C'est ce que réalise l'instruction SEC (Mise à 1 de la retenue). L'exécution se présente donc comme suit:

SEC	SEC →	code 38
LDA #13	LDA →	code A9
SBC #Ø8	SBC →	code E9
BRK	BRK →	code ØØ

L'adresse de début de programme sera également Ø1ØØ:

Touches pressées				**Affichage**		**Instruction**
				adresse	donnée	
RST	AD			xxxx	xx	
Ø	1	Ø	Ø	Ø1ØØ	xx	
DA		3	8	Ø1ØØ	38	SEC
+		A	9	Ø1Ø1	A9	LDA #
+		1	3	Ø1Ø2	13	
+		E	9	Ø1Ø3	E9	SBC #

+		Ø	8	Ø1Ø4	Ø8		
+		Ø	Ø	Ø1Ø5	ØØ		BRK
AD				Ø1Ø5	ØØ	⎫	
1	A	7	E	1A7E	xx	⎬	voir remarque
DA		Ø	Ø	1A7E	ØØ	⎬	
+		1	C	1A7F	1C	⎭	
AD							
Ø	1	Ø	Ø	Ø1ØØ	38		début de programme
GO				Ø1Ø7	xx		
AD				Ø1Ø7	xx		
Ø	Ø	F	3	ØØF3	ØB		résultat

Remarque Le chargement des adresses 1A7E et 1A7F remplit les mêmes fonctions que dans l'addition, à l'égard du sous-programme de sauvegarde du contenu des registres.
A l'adresse ØØF3, apparaît le résultat du programme de la soustraction; Ø8 soustrait de 13 (équivalent hexadécimal du nombre décimal 19) donne Ø8 (équivalent du nombre décimal 11).

Instructions logiques

Dans le cadre de l'adressage immédiat, nous examinerons encore un certain nombre d'instructions logiques.
L'instruction OR (OU) est bien connue. L'instruction ORA#(octet) a pour équation AVM→ A. Le code est Ø9. Servons-nous d'un petit programme pour suivre son exécution:

LDA#AA charger AA dans l'accumulateur
ORA#ØF exécuter la fonction OR entre AA et ØF, bit par bit
BRK arrêt
AA est le nombre binaire 1Ø1Ø1Ø1Ø
ØF est le nombre binaire ØØØØ1111
résultat 1Ø1Ø1111 (AF)

La fonction OR est exécutée pour les bits ayant même position. Le résultat est déposé dans l'accumulateur. La figure 2a présente le hardware (portes OR) équivalent.
La fonction AND (ET) est une autre opération logique. Elle a pour équation AΛM→ A. Le code est 29. Voici le programme:

LDA#AA charger AA dans l'accumulateur
AND#ØF exécuter la fonction AND entre AA et ØF, bit par bit
BRK arrêt
AA est le nombre binaire 1Ø1Ø1Ø1Ø
ØF est le nombre binaire ØØØØ1111
résultat ØØØØ1Ø1Ø (ØA)

La fonction AND est appliquée aux bits ayant même position. Le résultat est déposé dans l'accumulateur. La figure 2b présente le hardware (portes AND) équivalent.
Vient ensuite la fonction EOR (OU Exclusif). L'instruction EOR# a pour suite A∀M→ A. Le code est 49. Le programme est:

LDA#AA charger AA dans l'accumulateur

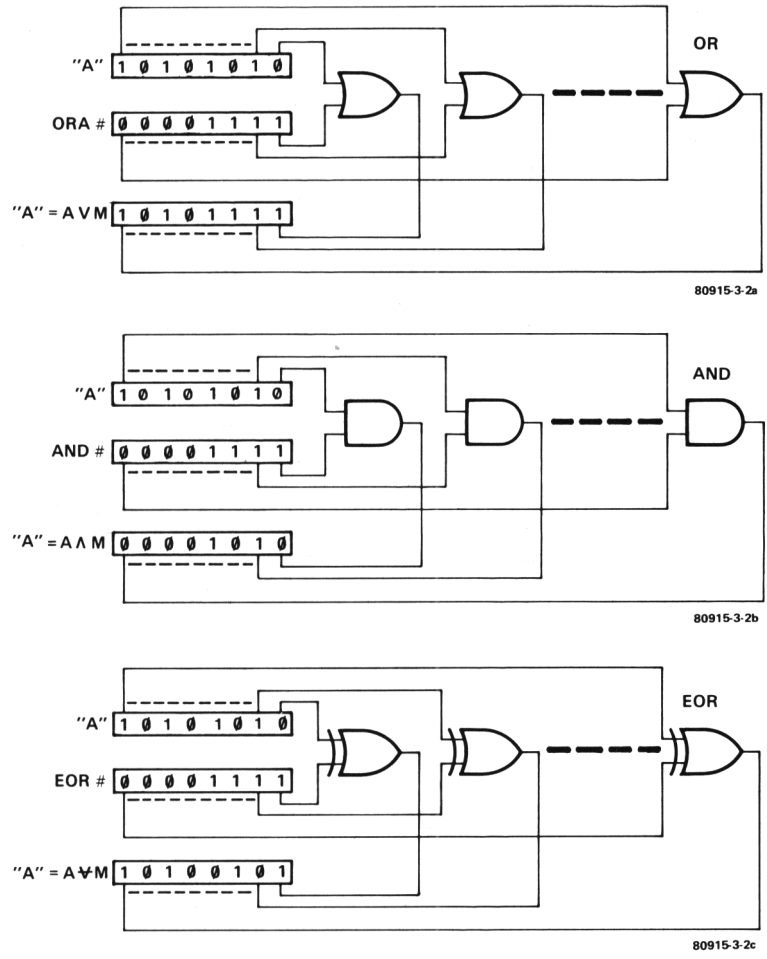


Figure 2. Représentation symbolique, grâce à des portes d'un type adapté, du traitement logique des opérations ORA (OU inclusif avec Accumulateur), AND (ET logique) et EOR (OU exclusif avec accumulateur).

EOR #ØF exécuter la fonction EOR entre AA et ØF, bit par bit
BRK arrêt
AA est le nombre binaire 1Ø1Ø1Ø1Ø
ØF est le nombre binaire ØØØØ1111
résultat 1Ø1ØØ1Ø1 (A5)
(si les deux entrées sont à 1, la sortie est Ø; si elles sont à Ø, la sortie est Ø, mais si l'une est à 1 et l'autre à Ø, la sortie est 1). La fonction EOR est exécutée pour les bits ayant même position. Le résultat est déposé dans l'accumulateur.

Les trois exemples précédents vont être réunis dans le programme ci-dessous:

Touches pressées				Affichage adresse	Affichage donnée	Instruction
RST	AD			xxxx	xx	
Ø	1	Ø	Ø	Ø1ØØ	xx	
DA		A	9	Ø1ØØ	A9	LDA # début 1
+		A	A	Ø1Ø1	AA	
+		Ø	9	Ø1Ø2	Ø9	ORA #
+		Ø	F	Ø1Ø3	ØF	
+		Ø	Ø	Ø1Ø4	ØØ	BRK fin 1
+		A	9	Ø1Ø5	A9	LDA # début 2
+		A	A	Ø1Ø6	AA	
+		2	9	Ø1Ø7	29	AND #
+		Ø	F	Ø1Ø8	ØF	
+		Ø	Ø	Ø1Ø9	ØØ	BRK fin 2
+		A	9	Ø1ØA	A9	LDA # début 3
+		A	A	Ø1ØB	AA	
+		4	9	Ø1ØC	49	EOR #
+		Ø	F	Ø1ØD	ØF	
+		Ø	Ø	Ø1ØE	ØØ	BRK fin 3
AD						
Ø	1	Ø	Ø	Ø1ØØ	A9	Adresse de début 1
GO				Ø1Ø6	AA	début 1
AD						
Ø	Ø	F	3	ØØF3	AF	résultat 1 fonction ORA
AD						
Ø	1	Ø	5	Ø1Ø5	A9	Adresse de début 2
GO				Ø1ØB	AA	début 2
AD						
Ø	Ø	F	3	ØØF3	ØA	résultat 2 fonction AND
AD						
Ø	1	Ø	A	Ø1ØA	A9	Adresse de début 3
GO				Ø11Ø	xx	début 3
AD						
Ø	Ø	F	3	ØØF3	A5	résultat 3 fonction EOR

L'intégration de trois instructions d'interruption software, BRK, nous a permis d'exécuter l'un après l'autre les trois sous-programmes. A chaque fois, il a suffi d'introduire l'adresse de début convenable, avant de presser la touche GO.
Les trois instructions logiques que nous venons d'étudier ont une importance considérable. Elles sont utilisées pour modifier un bit déterminé d'un

octet, dans que les autres bits soient transformés. A cet effet, l'octet objet de la modification est déposé dans l'accumulateur et la fonction OR, AND ou EOR est exécutée grâce à un masque (octet). Ce qui a pour conséquence que le bit à modifier est mis soit à "1", soit à "Ø", ou encore inversé.

Ainsi en avons-nous terminé avec l'adressage immédiat dont les instructions LDA#, LDX#, LDY#, ADC#, SBC#, ORA#, AND#, EOR# ont été étudiées. D'autre part, les instructions CLC, SEC et BRK ainsi que leurs adresses implicites seront explorées de manière plus détaillée.

Adressage absolu

Dans les adressages direct et immédiat, l'opérande suit immédiatement le code opération de l'instruction. Dans l'adressage absolu, cet opérande est, en fait, une adresse. L'instruction s'effectue donc sur le contenu de la mémoire désignée par cette adresse. Prenons une nouvelle fois l'exemple d'un programme pour illustrer ce qui vient d'être vu.

LDA-ØlØØ charger le contenu de l'emplacement mémoire ØlØØ dans l'accumulateur
STA-Ø15A écriture (copie) du contenu de l'accumulateur à l'emplacement mémoire Ø15A
LDA-ØlØl charger le contenu de l'emplacement mémoire ØlØl dans l'accumulateur
STA-Ø15B écriture du contenu de l'accumulateur à l'emplacement mémoire Ø15B
LDA-ØlØ2 charger le contenu de l'emplacement mémoire ØlØ2 dans l'accumulateur
STA-Ø15C écriture du contenu de l'accumulateur à l'emplacement mémoire Ø15C
LDA-ØlØ3 charger le contenu de l'emplacement mémoire ØlØ3 dans l'accumulateur
STA-Ø15D écriture du contenu de l'accumulateur à l'emplacement mémoire Ø15D
LDA-ØlØ4 charger le contenu de l'emplacement mémoire ØlØ4 dans l'accumulateur
STA-Ø15E écriture du contenu de l'accumulateur à l'emplacement mémoire Ø15E

Le tiret entre mnémomique et adresse caractérise l'adressage absolu.

Nous constatons que le contenu des emplacements ØlØØ . . . ØlØ4 a été copié aux emplacements Ø15A . . . Ø15E. L'accumulateur A a servi de station relais pour cette opération. Les quatre emplacements mémoire initiaux ont conservé leur contenu, d'où l'expression "copie". La figure 3 montre le contenu des emplacements mémoire ayant été copié.

Il n'est plus besoin de présenter l'instruction LDA; dans le cas présent le symbole # ne l'accompagne pas. Par contre, nous découvrons l'instruction STA-(STore Accumulator in Memory), c'est-à-dire, rangement du contenu de l'accumulateur dans la mémoire (n'oublions pas, qu'en fait, il s'agit du rangement de la copie du contenu, qui, lui-même, reste dans l'accumulateur). Cette opération est symbolisée comme suit: A→M. L'adressage absolu regroupe trois octets. Le premier octet livre le code de l'instruction (opcode); le second octet, qui est celui le plus à droite, est l'octet ADL

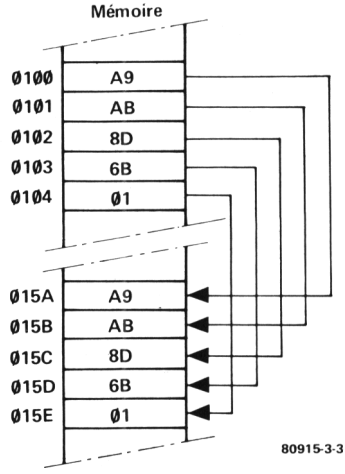


Figure 3. Sections de mémoire montrant la copie d'un petit programme en "adressage absolu".

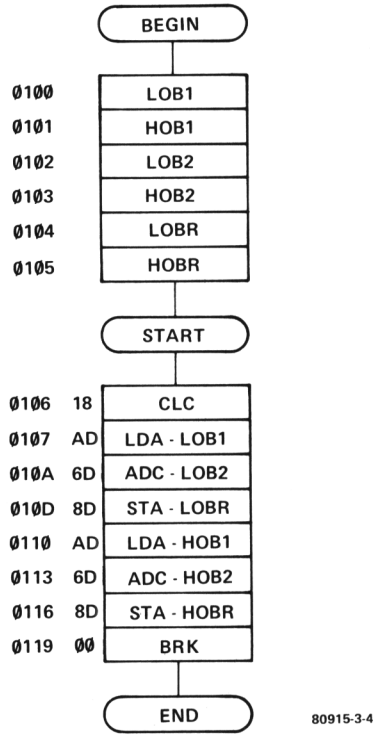


Figure 4. Organigramme, ou ordinogramme ("flow-chart") d'un programme d'addition de deux nombres de 16 bits.

(dans lequel L = Low = bas), le troisième, le plus à gauche, est l'octet ADH (H = High = haut).
Composons un programme à l'aide duquel deux nombres de 16 bits seront additionnés et dans lequel seront utilisées les instructions de l'adressage absolu. Le premier de ces nombres se compose d'un octet gauche HOB1 (High Order Byte) et d'un octet droit LOB1 (Low Order Byte); le second nombre est constitué de l'octet HOB2 et de l'octet LOB2. Le résultat sera un nombre à 16 bits, éventuellement avec retenue (carry) caractérisée par Ø1 ajouté à sa gauche, et partagé en un octet droit LOBR et un octet gauche HOBR.
Avant que nous exposions et commentions le programme lui-même, voyons d'abord l'organigramme en figure 4. Il nous permet de réserver six emplacements mémoire aux octets dont il a été question ci-dessus, à partir de l'emplacement ØlØØ, et qui serviront de matière première ou de produit fini du programme. Quant à celui-ci, il mobilisera les emplacements Ø11Ø6 à Ø119. Précisons encore, avant de commencer, que les deux nombres à additionner sont respectivement Ø4EF et 23AB.

Touches pressées					**Affichage** adresse	**Affichage** donnée	**Instruction**
(STEP: OFF; DISPLAY: ON)							
RST	AD				xxxx	XX	
1	A	7	A		1A7A	XX	
DA		Ø	Ø		1A7A	ØØ	Initialisation de STEP
+		1	C		1A7B	1C	
+ +					1A7D	XX	
+		Ø	Ø		1A7E	ØØ	Initialisation de BRK
+		1	C		1A7F	1C	
AD					1A7F	1C	
Ø	1	Ø	Ø		Ø1ØØ	XX	
DA		E	F		Ø1ØØ	EF	LOB1
+		Ø	4		Ø1Ø1	Ø4	HOB1
+		A	B		Ø1Ø2	AB	LOB2
+		2	3		Ø1Ø3	23	HOB2
+					Ø1Ø4	XX	emplacements pour LOBR
+					Ø1Ø5	XX	emplacements pour HOBR
+		1	8	CLC	Ø1Ø6	18	Flag Clear Carry
+		A	D	LDA-	Ø1Ø7	AD	
+		Ø	Ø		Ø1Ø8	ØØ	LOB1 dans Accumulateur
+		Ø	1		Ø1Ø9	Ø1	
+		6	D	ADC-	Ø1ØA	6D	
+		Ø	2		Ø1ØB	Ø2	A+LOB2→A (Flag Carry pris en compte)
+		Ø	1		Ø1ØC	Ø1	
+		8	D	STA-	Ø1ØD	8D	
+		Ø	4		Ø1ØE	Ø4	résultat (=LOBR) vers adresse Ø1Ø4
+		Ø	1		Ø1ØF	Ø1	

+		A	D	LDA-	Ø110	AD	HOB1 dans Accumulateur
+		Ø	1		Ø111	Ø1	
+		Ø	1		Ø112	Ø1	
+		6	D	ADC-	Ø113	6D	A+HOB2→A (Flag Carry pris en compte)
+		Ø	3		Ø114	Ø3	
+		Ø	1		Ø115	Ø1	
+		8	D	STA-	Ø116	8D	Résultat (=HOBR) vers adresse Ø1Ø5
+		Ø	5		Ø117	Ø5	
+		Ø	1		Ø118	Ø1	
+		Ø	Ø	BRK	Ø119	ØØ	Fin de programme
AD							
Ø	1	Ø	6		Ø1Ø6	18	Adresse de début
GO					Ø11B	xx	Boucle de programme
AD							
Ø	1	Ø	4		Ø1Ø4	9A	Résultat: LOBR
+					Ø1Ø5	28	Résultat: HOBR

Après avoir consacré tous ces instants à la frappe correcte des touches et à l'observation des afficheurs, prenons le temps de réfléchir aux opérations qui se sont déroulées. Les deux nombres hexadécimaux à additionner étaient:

1. nombre: Ø4EF, lequel s'écrit dans le système binaire
   ØØØØØ1ØØ 111Ø1111
   ←HOB1→ ←LOB1→
2. nombre: 23AB, lequel s'écrit dans le système binaire
   ØØ1ØØØ11 1Ø1Ø1Ø11
   ←HOB2→ ←LOB2→

Tout d'abord, le flag Carry chargé à l'adresse Ø1Ø6 est mis à "Ø". Après que LOB1 ait été chargé dans l'accumulateur et qu'il y ait été additionné à LOB2, le flag Carry est mis à 1:

```
          111Ø1111  LOB1
          1Ø1Ø1Ø11  LOB2
          111 1111  Carry
      + ----------
        1 1ØØ11Ø1Ø  LOBR
Carry     ←9→ ←A→
```

L'écriture de LOBR étant réalisée à l'adresse Ø1Ø4 et le chargement de HOB1 dans l'accumulateur étant opéré, le flag Carry ne change pas d'état. Continuons l'addition:

```
          ØØØØØ1ØØ  HOB1
          ØØ1ØØØ11  HOB2
                 1   Carry de LOBR (tout à fait à
                     droite maintenant!)
              111    Carry
      + ----------
        Ø ØØ1Ø1ØØØ   HOBR
Carry     2    8
```

Le résultat est écrit à l'adresse Ø1Ø5. Il suffit de taper les adresses Ø1Ø4 et Ø1Ø5 pour visualiser le résultat. D'autre part, chaque fois que le Junior Computer sera mis sous tension, la procédure d'initialisation demandera que les données ØØ et 1C soient chargées aux emplacements mémoire respectifs 1A7A et 1A7E (ØØ) ainsi que 1A7B et 1A7F (1C). Ces deux données, ØØ et 1C, forment un "vecteur" (1CØØ), notion sur laquelle nous reviendrons ultérieurement.

Le programme d'addition que nous avons détaillé ci-dessus faisait usage presque exclusif d'instructions dont l'adressage était absolu. En fait, on peut considérer que l'exécution d'une instruction comporte trois cycles dont chacun concerne respectivement le code opération (opcode), l'octet de poids faible (Low Order Byte) qui est aussi celui de droite, l'octet de poids fort (High Order Byte) qui est aussi celui de gauche.

Il existe bien d'autres instructions d'adressage absolu et nous en donnons un aperçu ci-après:

Chargement et écriture:

LDA- code AD (M→ A) charger le contenu de la mémoire dans A
LDX- code AE (M→ X) charger le contenu de la mémoire dans X
LDY- code AC (M→ Y) charger le contenu de la mémoire dans Y
STA- code 8D (A→ M) rangement de l'accumulateur en mémoire
STX- code 8E (X→ M) rangement de X en mémoire
STY- code 8C (Y→ M) rangement de Y en mémoire

Instructions arithmétiques:

ADC- code 6D (A+M+C→ A) ajouter mémoire à accumulateur avec Carry
SBC- code ED (A-M-$\overline{C}$→ A) soustraire mémoire de l'accumulateur avec $\overline{Carry}$
INC- code EE (M+1→ M) incrémenter la mémoire de 1
DEC- code CE (M-1→ M) décrémenter la mémoire de 1

Ces deux dernières instructions permettent d'augmenter (INC) ou de diminuer (DEC) le contenu de la mémoire de 1 (ØØØØØØØ1).

Instructions logiques:

ORA- code ØD (A∨M→ A) OU inclusif avec l'accumulateur
AND- code 2D (A∧M→ A) ET logique avec l'accumulateur
EOR- code 4D (A∀M→ A) OU exclusif avec l'accumulateur

Nous avons déjà rencontré ces trois instructions, mais avec le symbole # au lieu de —.

Exécution pas-à-pas

Une pression sur la touche GO après que les différentes instructions et l'adresse de début aient été introduites à l'aide du clavier a pour conséquence le lancement du programme et son déroulement jusqu'à ce qu'il rencontre une instruction BRK. C'est le cas lorsque le commutateur STEP est à la position OFF. Quand nous le portons à la position ON, la LED rouge intégrée dans la touche STEP/GO s'illumine.

Si nous avons pris la précaution d'introduire la donnée ØØ à l'adresse 1A7A et la donnée 1C à l'adresse 1A7B avant le lancement du programme, celui-ci va se dérouler pas-à-pas. Notons bien que nous pressons d'abord la touche GO, que nous positionnons le commutateur STEP sur ON, et

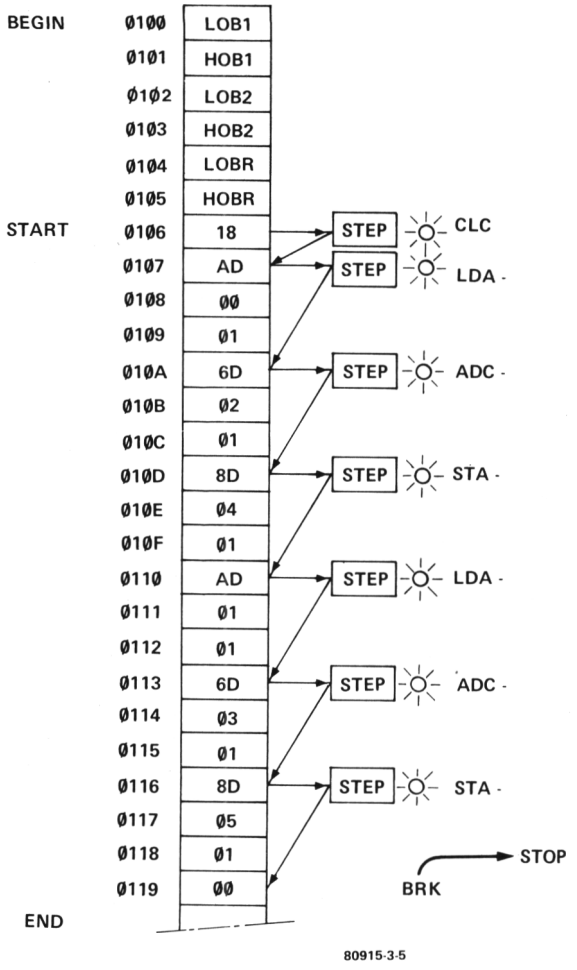
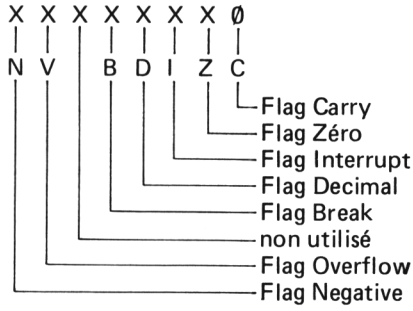


Figure 5. Le programme de la figure 4 représenté en carte mémoire.

qu'enfin nous pressons la touche STEP (qui est la même que la touche GO), la LED étant illuminée.
Désormais, chaque fois que nous presserons la touche STEP, une instruction sera exécutée, ainsi que le montre la figure 5 qui nous présente une partie de la mémoire lors du déroulement du programme avec "adressage absolu".
Imaginons donc que nous soyons arrivés à l'adresse **Ø1Ø6** et que nous pressions la touche STEP. Le flag Carry (CLC) est mis à "Ø". Le programme stoppe. Mais, nous aimerions bien savoir si le flag a été effectivement mis à "Ø". Souvenons-nous que l'état de tous les flags (indicateurs) est consigné dans le registre P et que le contenu de ce registre d'état est

sauvegardé à l'adresse ØØF1. Le plus simple est donc d'afficher le contenu de la mémoire à cette adresse. Par conséquent, nous pressons les touches AD, Ø, Ø, F et 1. Examinons l'affichage. Les afficheurs d'adresse font bien apparaître l'adresse ØØF1. Mais, qu'en est-il des afficheurs de donnée?
Eh bien, nous ne pouvons connaître le contenu de cette adresse que si le flag Carry a réellement été remis à "Ø" et il faut que nous connaissions la composition de l'octet à l'adresse ØØF1; après visualisation de celle-ci nous constatons que la composition de l'octet est la suivante:



Les "X" signifient que les bits peuvent être à "1" ou à "Ø". Leur état dépend de ce qu'il était au moment où le Junior Computer a été initialisé. En tout état de cause, le bit de poids le plus faible est le bit de Carry. Lorsqu'il est à "Ø" (après une instruction CLC), le chiffre hexadécimal le plus à droite apparaissant sur les afficheurs des données doit égaler un nombre décodé décimal pair. Si le flag Carry est à "1", le nombre décimal est impair. Cette fois, les afficheurs de données ont fourni la réponse que nous attendions, et nous constatons qu'il nous est possible de contrôler l'exécution de l'instruction CLC.
La pression sur la touche AD nous a permis de faire une sorte d'excursion et nous voudrions revenir à l'adresse où l'instruction suivante (après STEP) attend d'être exécutée. Nous allons réaliser cela grâce à une pression sur la touche PC (Program Counter), car, nous le savons, c'est dans le compteur ordinal que se trouve stockée l'adresse de la prochaine instruction à exécuter.
Sitôt après la pression sur la touche PC, les afficheurs font apparaître Ø107 AD. AD est le code opération (opcode) de l'instruction LDA-(Chargement de l'accumulateur, pour l'adressage absolu). Nous pressons la touche STEP et les afficheurs nous indiquent Ø10A 6D. Dans le même temps, LDA-, l'instruction précédente a été exécutée, et, par conséquent, L0B1 (soit EF) devrait avoir été chargé dans l'accumulateur. Nous vérifions: le contenu de A (Accumulateur) est sauvegardé à l'adresse ØØF3. Pressons les touches AD, Ø, Ø, F et 3; c'est bien exact, l'affichage nous indique ØØF3 EF. Une légère pression sur PC, et nous pouvons continuer notre contrôle pas-à-pas, ou reprendre le déroulement continu du programme, tout simplement parce qu'une vérification complète n'est plus nécessaire ou souhaitable. Dans ce cas, le commutateur STEP est remis en position OFF.
Constatons que la procédure du pas-à-pas est un atout puissant dans la

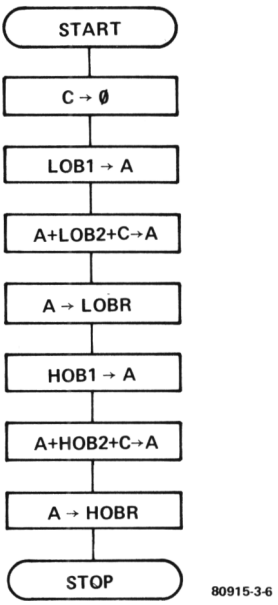


Figure 6. Ordinogramme "sommaire" du programme de la figure 4.

lutte contre les erreurs de programmation (elle permet de réussir plus facilement le "débugging" ou "déverminage").

Un peu d'organisation interne

Dorénavant, nous présenterons tous les programmes-types sous trois formes: 1. le programme sera décrit en quelques mots et accompagné d'un ordinogramme sommaire; voyez, par exemple, le programme d'addition de la figure 6. 2. un ordinogramme affiné accompagné des codes opération; reportez-vous au programme d'addition de la figure 4. 3. le tableau des touches à presser et l'affichage résultant.

Adressage en page-zéro

Il s'agit d'une forme particulière de l'adressage absolu. En ce qui concerne celui-ci, il demande trois octets: le premier est le code opération, le second est l'octet d'adresse de poids faible (ADL) qui se place à droite, et le troisième est l'octet de poids fort (ADH) qui se place à gauche. L'adressage page-zéro se caractérise par le fait que l'octet ADH est toujours nul (ØØ). Par conséquent, toutes les adresses allant de ØØØØ . . . ØØFF sont utilisables. Ces 256 adresses font partie de ce que nous appelons la page zéro. Les 256 adresses suivantes, allant de Ø1ØØ à Ø1FF appartiennent à la page 1, et il en va ainsi jusqu'à la page FF, qui comporte les adresses FFØØ . . . FFFF, grâce à quoi nous constatons que la totalité des adresses,

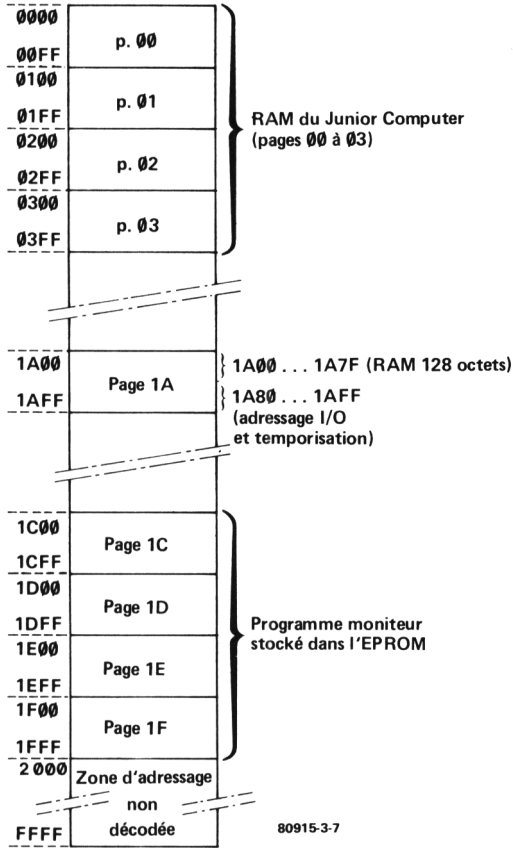


Figure 7. La mémoire est divisée en pages de chacune 256 octets. A l'intérieur d'une page, les deux "bits hexadécimaux" de gauche sont égaux; les deux bits hexadécimaux de droite varient de 00 à FF.

soit 256^2 = 65 536, sont réparties sur 256 pages comportant chacune 256 adresses.

La figure 7 illustre ce qu'est la pagination par la présentation d'une carte de la mémoire. Les pages 0 à 3 correspondent à la RAM (ou mémoire de travail). La page 1A concerne le PIA et correspond à la RAM interne ainsi qu'à l'adressage I/O. Les pages 1C . . . 1 F sont réservées au programme moniteur. Dans la version standard du Junior Computer, la limitation de la capacité de décodage (voir le chapitre 1) impose que 1FFF soit l'adresse la plus haute exploitable.

Comme l'ordinateur, grâce au code opération, sait automatiquement qu'une instruction de page-zéro est caractérisée par le fait que l'octet le plus à gauche est nul, il ne sera plus nécessaire de préciser ce point. Cela a pour effet de permettre l'utilisation de deux octets au lieu de

trois et d'économiser 1/3 des emplacements mémoire! Les instructions de page-zéro sont donc composées de deux octets: le code opération suivi d'un octet renfermant la partie la plus à droite de l'adresse (ADL). Les mnémoniques du code opération sont souvent complétés par un Z, de zéro, bien que ce ne soit pas obligatoire.
Ecriture et lecture:
LDAZ code A5 (M → A) charger le contenu de la mémoire dans A
LDXZ code A6 (M → X) charger le contenu de la mémoire dans X
LDYZ code A4 (M → Y) charger le contenu de la mémoire dans Y
STAZ code 85 (A → M) ranger l'accumulateur dans le mémoire
STXZ code 86 (X → M) ranger index X dans la mémoire
STYZ code 84 (Y → M) ranger index Y dans la mémoire

Instructions arithmétiques:
ADCZ code 65 (A+M+C → A) ajouter mémoire à accu. avec carry
SBCZ code E5 (A-M-C → A) soustraire mémoire de l'accu. avec carry
INCZ code E6 (M+1 → M) incrémenter la mémoire de 1
DECZ code C6 (M-1 → M) décrémenter la mémoire de 1

Instructions logiques:
ORAZ code Ø5 (A∨M → A) OU inclusif avec l'accumulateur
ANDZ code 25 (A∧M → A) ET logique avec l'accumulateur
EORZ code 45 (A∀M → A) OU exclusif avec l'accumulateur

Adressage relatif

Cette méthode d'adressage est employée pour l'exécution de sauts ou instructions de branchement. Il en existe de deux sortes, conditionnelles et inconditionnelles. En ce qui concerne les premières, le déplacement est exécuté sous certaines conditions, pour les secondes le branchement est toujours effectué. Les instructions du branchement conditionnel sont représentées dans les organigrammes par un losange (voir chapitre 2) et se traduisent dans le programme de travail par des flags de test qui indiquent la condition: si le flag est à "Ø", faire ceci, s'il est à "1", faire cela.
Voici les instructions du branchement conditionnel:

1. BCC, code 9Ø: Branchement si pas de retenue. Saut si C = Ø (reset)
   BCS, code BØ: Branchement si retenue. Saut si C = 1 (set)
2. BNE, code DØ: Branchement si non égal à Ø. Saut si Z = Ø (reset)
   BEQ, code FØ: Branchement si égalité. Saut si Z = 1 (set)
3. BPL, code 1Ø: Branchement si plus. Saut si N = Ø (reset)
   BMI, code 3Ø: Branchement à moins. Saut si N = 1 (set)
4. BVC, code 5Ø: Branchement si pas de débordement. Saut si V = Ø (reset)
   BVS, code 7Ø: Branchement si débordement. Saut si V = 1 (set)

Ces instructions nous permettrons de rédiger des programmes comportant des branchement inconditionnel seront étudiées ultérieurement.
Examinons d'abord le programme dont l'organigramme est présenté en figure 8. Il débute à l'adresse Ø2ØØ avec l'instruction LDY ØA, ce qui

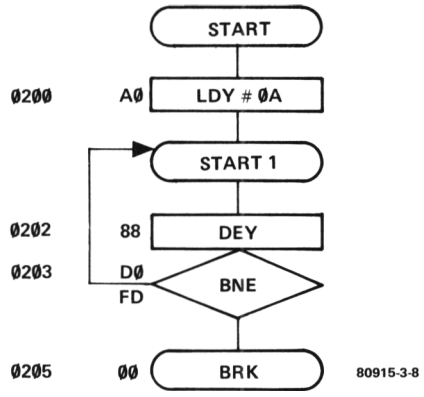


Figure 8. Ordinogramme d'un programme comportant une instruction de branchement conditionnel. Les codes opération et le déplacement sont indiqués.

veut dire qu'il faut charger le nombre hexadécimal ØA dans le registre Y du microprocesseur. La seconde instruction est DEY (décrémenter Y), grâce à quoi le contenu du registre Y sera diminué de 1. Après l'exécution de cette opération, le contenu de Y est devenu Ø9. L'instruction suivante est BNE qui va permettre de tester l'état du bit Z du registre d'état. Le flag zéro sera mis à 1 lorsque le contenu du registre Y sera nul. Mais, le contenu de celui-ci n'est pas nul, puisque le mot Ø9 y a été chargé et le flag zéro est donc nul. Par conséquent, le programme va effectuer la boucle BNE jusqu'à ce que le contenu de Y soit devenu nul. Dès que ce sera le cas, la condition impliquant le branchement ne sera plus remplie et le programme stoppera à l'instruction suivante BRK. Le Junior Computer reviendra au moniteur (adresse 1CØØ).
La figure 8 nous montre, à gauche du losange symbolisant le branchement, le code opératoire de l'instruction BNE, DØ, ainsi que FD. Ce nombre hexadécimal indique le nombre d'emplacements mémoire correspondant au déplacement en avant (s'il est positif), en arrière (s'il est négatif) qui doit être effectué pour atteindre l'instruction DEY indiquée par la flèche qui part du losange et va jusqu'à START 1.
Le mot FD correspond au nombre binaire 11111ØØ, lequel est le complément à deux de - 3. Autrement dit, le microprocesseur doit revenir de trois emplacements en arrière pour passer de l'instruction BNE à l'instruction DEY, ce qui s'opère comme suit:

```
Ø2ØØ    AØ   LDY #
Ø2Ø1    ØA
Ø2Ø2    88   DEY
Ø2Ø3    DØ   BNE
Ø2Ø4    FD       déplacement à effectuer
Ø2Ø5    BRK
```

Immédiatement après l'exécution de l'instruction correspondant à l'adresse Ø2Ø4, le compteur ordinal, PC, pointe déjà vers l'adresse suivante Ø2Ø5.

Il indique l'emplacement mémoire dans lequel est contenu le *déplacement,* lequel est un nombre décimal précisant le nombre de pas à effectuer en avant ou en arrière.
Une instruction de branchement relatif autorise un déplacement de 127 pas en avant et de 128 pas en arrière, soit un total de 256 octets. Pour calculer sa valeur, il faut partir de l'état du compteur ordinal, ou, plus pratiquement, de l'adresse de l'instruction faisant suite à l'instruction de branchement conditionnel. Sans expédient, ce n'est pas une mince affaire.

Calcul du déplacement à l'aide du moniteur

Pour exécuter une instruction de branchement, il est nécessaire de savoir où le branchement commence (origine) et où il se termine. Mais, il est clair qu'avec l'établissement d'un organigramme à l'exemple de la figure 8, ces deux adresses sont connues, car, dans le cas contraire, il ne serait pas possible de faire effectuer le calcul par l'ordinateur.
En conservant l'exemple de la figure 8, voici comment cela se passe:
D'abord on tape l'adresse 1FD5 qui est l'adresse de départ du programme de calcul du déplacement, contenu dans le moniteur du Junior Computer. Ensuite, on presse la touche GO, puis on tape l'octet le plus à droite de l'adresse où est stocké le code opération de l'instruction de branchement, Ø3, et immédiatement après, l'octet le plus à droite de l'adresse finale, Ø2. Ces deux mots apparaissent sur les afficheurs d'adresse, les afficheurs de donnée faisant apparaître le déplacement, FD:

```
AD                  XXXX  XX
1    F    D    5     1FD5  D8
GO                  1FD5  4Ø
Ø    3    Ø    2     Ø3Ø2  FD  noter le déplacement!
RST                 Ø3Ø2  XX
```

Après enfoncement de la touche RST, l'ordinateur abandonne le programme de calcul du déplacement. Il est alors possible de taper le programme de la figure 8:

```
AD
Ø    2    Ø    Ø     Ø2ØØ  XX
DA        A    Ø     Ø2ØØ  AØ  LDY #
+         Ø    A     Ø2Ø1  ØA
+         8    8     Ø2Ø2  88  DEY
+         D    Ø     Ø2Ø3  DØ  BNE
+         F    D     Ø2Ø4  FD  déplacement
+         Ø    Ø     Ø2Ø5  ØØ  BRK
```

Ensuite l'adresse de début est introduite et le programme est lancé en pressant l'une des touches GO ou STEP. Qu'il ne soit indispensable de préciser que l'utilisation du seul octet de poids faible de l'adresse de début et de l'adresse de fin de branchement, résulte du fait que les possibilités de déplacement s'élèvent à un maximum de 256. Cela équivaut au contenu d'un octet, ou l'équivalent de deux caractères hexadécimaux.

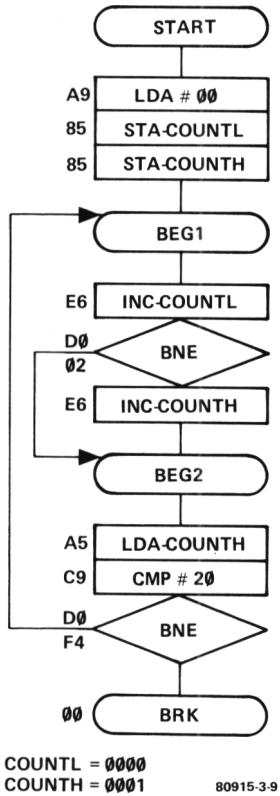


Figure 9. Ordinogramme d'un programme compteur ("software-teller").

Un compteur software

Nous allons maintenant aborder un programme grâce auquel il est possible de réaliser un compteur et qui servira d'illustration complémentaire aux instructions de branchement conditionnel, sans modification matérielle. Par la même occasion, nous ferons connaissance avec une nouvelle instruction, CMP, qui veut dire comparer à l'accumulateur.

Il s'agit d'un compteur de nombres hexadécimaux allant de l'adresse ØØØØ à l'adresse 2ØØØ et qui s'arrête de compter quand la dernière est atteinte. La figure 9 en présente l'organigramme. Le nombre 2ØØØ est formé de 16 bits; il est nécessaire de mobiliser deux emplacements mémoire pour déterminer l'état du compteur. L'emplacement COUNTH (adresse ØØØ1) contient l'octet gauche (de poids fort), et l'emplacement COUNTL (adresse ØØØØ) l'octet droit (de poids faible).

On commence par charger ØØ dans l'accumulateur et l'on met le compteur à zéro, via l'accumulateur. Après l'achèvement de la procédure de remise à zéro, le contenu de l'emplacement COUNTL est incrémenté de 1, et

nous en arrivons à la première instruction de branchement NBE, grâce à laquelle on pourra savoir si le contenu de COUNTL est nul ou non, ce qui équivaut à tester l'état du flag Z. Le contenu de l'emplacement mémoire étant différent de zéro, un saut est effectué jusqu'à BEG2 et suivantes.
Le contenu de COUNTH est chargé dans l'accumulateur et le contenu de celui-ci est ensuite comparé à la valeur finale, 2Ø, par l'intermédiaire de l'instruction CMP. Celle-ci a pour effet d'influencer l'état du flag Z (code opération de CMP = C9). Celui-ci est mis à 1 lorsque le contenu de l'accumulateur, et donc de COUNTH, est égal à 2Ø, et il reste à Ø tant que ce n'est pas le cas. Quand le contenu de COUNTH est égal à 2Ø, la condition nécessaire pour qu'un branchement ait lieu avec la seconde instruction BNE n'est plus remplie. Le microprocesseur quitte la boucle de comptage et arrive à l'instruction BRK qui achève le programme.
En d'autres termes, tant que le contenu de COUNTH est inférieur à 2Ø, le compteur software incrémente. Voici comment le programme est introduit grâce au clavier du Junior Computer:

Touches pressées					**Affichage adresse**		**donnée**	
AD					xxxx		xx	
1	F	D	5		1FD5		D8	adresse de départ du calcul de déplacement
GO					ØØØØ		ØØ	
1	8	1	C		181C		Ø2	premier déplacement
2	Ø	1	6		2Ø16		F4	second déplacement
RST	AD							
0	2	1	Ø		021Ø		xx	adresse du début de programme
DA		A	9		ØØ21Ø		A9	LDA #
+		Ø	Ø		Ø211		ØØ	
+		8	5		Ø212		85	STA-
+		Ø	Ø		Ø213		ØØ	
+		8	5		Ø214		85	STA-
+		Ø	1		Ø215		Ø1	
+		E	6		Ø216		E6	INC-
+		Ø	Ø	12	Ø217		ØØ	
+		D	Ø	11	Ø218		DØ	BNE
+		Ø	2	10	Ø219		Ø2	déplacement
+		E	6	9	Ø21A		E6	INC-
+		Ø	1	8	Ø21B	1	Ø1	
+		A	5	7	Ø21C	2	A5	LDA-
+		Ø	1	6	Ø21D		Ø1	
+		C	9	5	Ø21E		C9	CMP #
+		2	Ø	4	Ø21F		2Ø	opérande CMP #
+		D	Ø	3	Ø22Ø		DØ	BNE
+		F	4	2	Ø221		F4	déplacement

+		Ø	Ø	1	Ø222	ØØ	BRK
AD					Ø222	xx	
Ø	2	1	Ø		Ø21Ø	A9	adresse de début
GO					Ø212	xx	début de programme

Les branchements (du compteur de programme) à effectuer éventuellement, après une BNE, sont indiqués dans le programme. Il y en a 12 négatifs et deux positifs. Les déplacements sont F4 ou Ø2.
Examinons d'un peu plus près l'instruction CMP; elle sert à comparer le contenu de l'accumulateur à des données contenues dans la mémoire. En cas d'adressage immédiat (CMP #, code opération C9), les deux octets qui suivent indiquent l'adresse où est stockée la donnée grâce à laquelle la comparaison peut être faite. Il existe d'autres possibilités d'adressage de l'instruction CMP.
La donnée est soustraite du contenu de l'accumulateur (A-M); le résultat n'est pas rangé dans A, mais il influence l'état des flags du registre d'état du microprocesseur. Le contenu de l'accumulateur n'est pas modifié. Le résultat de A-M est nul ou non nul. A est égal à M, ou il lui est inégal.
Trois flags sont influencés par le résultat:

a) le flag N est mis à 1 lorsque A-M est négatif
   à Ø lorsque A-M est positif
   (souvenons-nous que pour les nombres négatifs, le bit le plus à gauche est 1, pour les nombres positifs c'est Ø)
b) le flag Z est mis à 1 lorsque A-M est nul
   à Ø lorsque A-M est non nul
c) le flag C est mis à 1 lorsque $M \leqslant A$
   à Ø lorsque $M > A$

Pour nous résumer, l'étude des instructions du branchement conditionnel nous a appris trois choses:

a) L'instruction CMP nous permet de préciser si le contenu de l'accumulateur est ou n'est pas plus petit qu'une donnée déterminée; l'instruction de branchement BCC permet, qu'en fonction du résultat, l'on sache si un branchement peut être effectué ou non.
b) Après l'exécution d'une instruction CMP, on sait si le contenu de l'accumulateur est égal ou non à une donnée déterminée; l'instruction de branchement BEQ autorise, en fonction du résultat de la comparaison, que l'on sache si le branchement est effectué ou non.
c) Après l'exécution d'une instruction CMP, on sait si le contenu de A est supérieur ou égal à une donnée déterminée; l'instruction de branchement BCS permet de découvrir si, en fonction du résultat, le branchement sera effectué ou non.

Finissons-en provisoirement avec l'étude des instructions du branchement conditionnel.

Instructions du branchement inconditionnel

Les instructions de branchement inconditionnel permettent de faire un saut jusqu'à une adresse à partir de laquelle le programme se poursuit. Pour l'exécution d'un tel saut, le 6502 dispose d'une instruction très pratique, *JMP*, ou saut à une adresse. C'est une instruction de branchement inconditionnel dont l'adressage est absolu ou indirect. Le code opé-

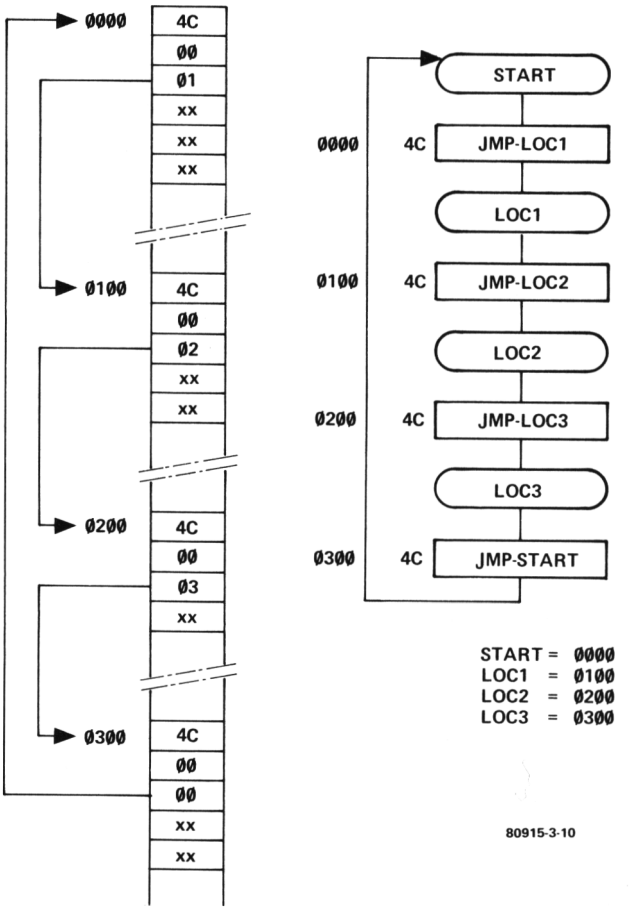


Figure 10. Programme de longueur théoriquement illimitée comportant quatre instructions de saut inconditionnel.

ration de l'adressage absolu est 4C, celui de l'adressage indirect est 6C. Le format de l'instruction est donc de trois octets, dont les deux derniers contiennent l'adresse où se fera le saut. La figure 10 montre l'organigramme et les diverses étapes du programme comportant l'exécution d'instructions JMP. A partir de START, quatre sauts seront effectués consécutivement à quatre adresses.

Les instructions JSR et RTS

L'instruction JSR, ou saut à un sous-programme, permet de passer à un sous-programme qui remplit une fonction spécifique, par exemple, la commande d'un affichage à 7 segments, la traduction d'un nombre hexa-

décimal dans le code ASCII correspondant ou le transport de données d'une section de mémoire dans une autre. C'est un programme auxiliaire mobilisé pour les besoins de l'exécution du programme principal. Pour le quitter, on se sert de l'instruction RTS (ReTurn from Subroutine) ou retour de sous-programme. Le microprocesseur revient au programme principal, et très précisément à l'instruction qui suit immédiatement l'instruction JSR ayant initialisé le saut au sous-programme.

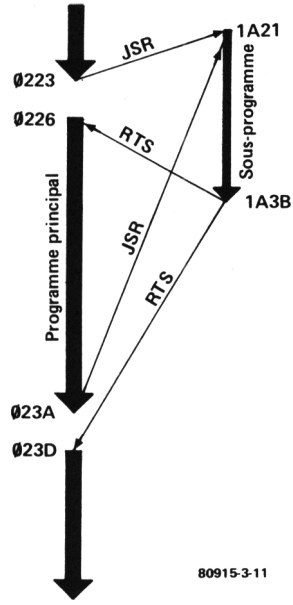


Figure 11. Illustration d'un sous-programme auquel il est fait deux fois appel.

L'un des avantages des sous-programmes commandés par les instructions JSR et RTS est qu'ils permettent d'économiser un grand nombre d'emplacements mémoire, en fonction de la fréquence des appels qui leur sont faits. En principe, il n'y a pas de limite dans ce domaine. D'autre part, la structure du programme principal reste beaucoup plus claire.
Pour compléter notre compréhension du mode d'utilisation de ces instructions, ayons recours au programme de la figure 11 où nous voyons un sous-programme auquel il est fait appel deux fois. Il occupe les emplacements mémoire allant de 1A21 à 1A3B. Le code opération d'un instruction JSR est stocké à l'adresse Ø223, tandis que l'adresse du début du sous-programme est mémorisée aux emplacements Ø224 (ADL) et Ø225 (ADH).
Cette dernière adresse est contenue dans une pile, dont nous examinerons prochainement la structure, et elle est la dernière du programme principal avant que soit exécutée l'instruction de saut au sous-programme. La seconde instruction JSR est notifiée à l'adresse Ø23A; les adresses Ø23B

et Ø23C contiennent les mêmes données que les adresses Ø224 et Ø225. Une instruction RTS provoque la reprise du programme principal à partir de l'adresse Ø23D.

Pile et pointeur de pile

Jusqu'à présent, nous avons admis tacitement que le microprocesseur retrouvait de lui-même l'adresse voulue après qu'il ait quitté un sous-programme pour reprendre le déroulement du programme principal, à la suite d'une instruction RTS. Mais, le moment est venu d'examiner de plus près le mécanisme assurant cette fonction.

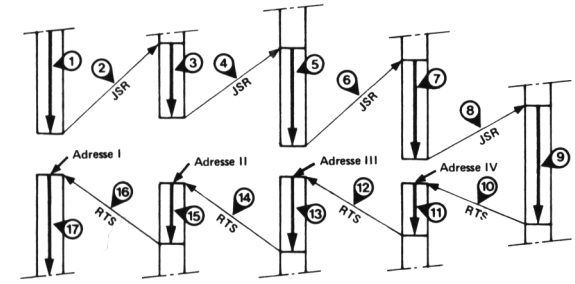
Chaque saut à un sous-programme implique un retour au programme principal et à chaque adresse d'une instruction de saut correspond une adresse de retour au programme principal. Le Junior Computer doit donc gérer les adresses de retour et il le fait grâce à une *pile*. C'est une section de la mémoire dans laquelle les adresses de retour (à raison de deux octets par adresse) sont empilées les unes sur les autres, dans un ordre déterminé formant une structure LIFO (Last In, First Out). Cela signifie tout simplement que ces adresses constituant une pile, la dernière adresse déposée au sommet de la pile est la première à y être prélevée. Imaginons que l'on empile des assiettes pour les laver, ce sera la dernière posée au sommet qui sera prise la première pour être lavée.

Notons également qu'il existe la structure FIFO (First In, First Out) grâce à laquelle les adresses sont arrangées en *file* afin que la première entrée soit aussi la première sortie.

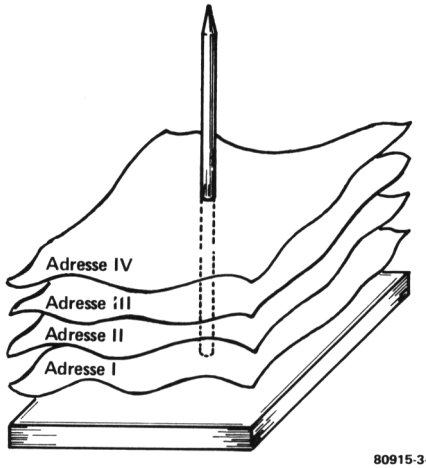
La figure 12 illustre la constitution d'une pile de sous-programmes sous la forme d'une liasse d'adresses constituant une structure LIFO. Le traitement s'effectue dans l'ordre 1 . . . 17 (voir figure 12a), le sous-programme 3 étant exécuté avant le sous-programme 2, celui-ci l'étant avant le sous-programme 1, avec, finalement, retour au programme principal, tout ceci en fonction de l'ordre des entrées des adresses (dernière adresse entrée: IV, premier sous-programme exécuté: 3). Le mécanisme d'empilage est très clairement illustré par la figure 12b, où l'on voit que toute feuille enfilée sur le support correspond à l'exécution d'un saut vers un sous-programme; dès que celui-ci est achevé, la feuille est enlevée et c'est la feuille qui se trouvait en-dessous d'elle qui prend sa place sur le dessus de la pile. En outre, l'opération représentée par la figure 12a n'est qu'une des innombrables possibilités existantes.

Les 256 emplacements mémoire de la page 1, dont les adresses vont de Ø1FF à Ø1ØØ, peuvent former une pile comportant un maximum de 128 adresses de retour. Sur une carte mémoire, les adresses sont disposées du haut vers le bas, selon un ordre croissant. Les emplacements d'une pile sont chargés du bas vers le haut (voir figure 12b), les premières adresses de retour étant Ø1FF et Ø1FE.

La figure 12c montre la position relative de la pile et du pointeur de pile à huit instants différents. Nous constatons que l'octet de gauche, ADH, de l'adresse de retour est toujours le premier chargé, et, qu'ensuite vient le chargement de l'octet de droite ADL. Nous voyons également que le pointeur pointe vers le dernier emplacement vide de la pile. Ce que ne nous montre pas la figure 12c, c'est que l'adresse de retour est constituée par l'adresse du programme principal où est stocké l'octet de gauche de



80915-3-12a



Pointe à mémos

80915-3-12b

Ø1 F9	Ø1F8	XX
	Ø1F9	XX
	Ø1FA	ADHIII
	Ø1FB	ADLIII
	Ø1FC	ADHII
	Ø1FD	ADLII
	Ø1FE	ADHI
	Ø1FF	ADLI

après (6) et (12)

Ø1 F7	Ø1F7	XX
	Ø1F8	ADHIV
	Ø1F9	ADLIV
	Ø1FA	ADHIII
	Ø1FB	ADLIII
	Ø1FC	ADHII
	Ø1FD	ADLII
	Ø1FE	ADHI
	Ø1FF	ADLI

après (8) et (10)

Ø1 FB	Ø1F8	XX
	Ø1F9	XX
	Ø1FA	XX
	Ø1FB	XX
	Ø1FC	ADHII
	Ø1FD	ADLII
	Ø1FE	ADHI
	Ø1FF	ADLI

après (4) et (14)

Ø1 FD	Ø1F8	XX
	Ø1F9	XX
	Ø1FA	XX
	Ø1FB	XX
	Ø1FC	XX
	Ø1FD	XX
	Ø1FE	ADHI
	Ø1FF	ADLI

après (2) et (16)

80915-3-12c

Figure 12. L'imbrication des sous-programmes (a). L'état de la pile (stack) rappelle celui d'une pointe à mémos (b). L'état de la pile et du pointeur de pile (stack pointer) en huit instants donnés, représentés également en figure "a" (c).

l'adresse de début du sous-programme concerné (ADH). Par contre, la figure 13 nous l'indique très clairement.

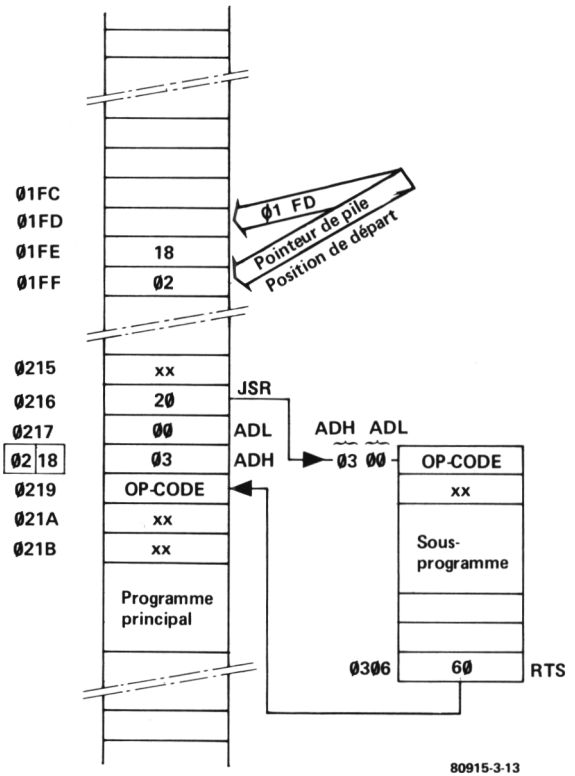


Figure 13. Exemple d'un programme comportant un sous-programme.

Ce programme est précédé d'un autre qui s'achève à l'adresse 0215. Ensuite, tout commence à l'adresse 0216 où est stocké le code opération 20, de l'instruction JSR. Mais, où va conduire cette instruction de saut? Le contenu des adresses 0217 et 0218 va l'indiquer, car, à la première est emmagasiné l'octet droit, 00, et à la seconde, l'octet gauche, 03, de l'adresse de début du sous-programme. Dans la pile, à la page 1 de la mémoire, sont stockés à l'adresse 01FF, l'octet gauche, 02, et à l'adresse 01FE, l'octet droit, 18, qui constituent l'adresse de la dernière opération entièrement achevée avant que le saut ne soit exécuté (en fait, c'est une partie de l'opérande de JSR).
Lorsque le sous-programme aura été exécuté et que le saut de retour sera effectué, le programme principal reprendra son déroulement à l'adresse qui suit l'adresse de la dernière opération complètement achevée avant que le saut ne soit effectué. Le sous-programme commence à l'adresse 0300 où est stocké le code opération de la première instruction. Bien

entendu, d'autres opérations suivent, mais, il n'est pas important que nous les abordions de manière précise. Quoi qu'il en soit, le sous-programme finit par arriver à l'adresse Ø3Ø6 dont le contenu est le code opération de l'instruction RTS qui ne comporte qu'un octet dans le cas d'adressage implicite (sur lequel nous reviendrons). En l'occurrence, l'expression "implicite" signifie que l'opérande de RTS (grâce à laquelle le saut de retour s'opérera) est située à la partie supérieure de la pile. Le retour au programme principal se fait à l'adresse Ø219 où est stocké le code opération d'une nouvelle instruction.

Du bon usage des touches

Il est temps que nous nous intéressions à un programme dans lequel les instructions de saut, dont nous avons tant parlé, seront utilisées. En même temps, nous nous familiariserons avec quelques "astuces" et ferons connaissance avec le riche répertoire du moniteur.

Nous allons donc choisir un programme permettant de faire apparaître sur les afficheurs du Junior Computer les nombres hexadécimaux correspondant à chaque touche du clavier. Le tableau ci-dessous nous les fait connaître:

Ø : ØØ	5 : Ø5	A : ØA	F : ØF	PC : 14
1 : Ø1	6 : Ø6	B : ØB	AD: 1Ø	
2 : Ø2	7 : Ø7	C : ØC	DA: 11	
3 : Ø3	8 : Ø8	D : ØD	+ : 12	
4 : Ø4	9 : Ø9	E : ØE	GO: 13	

L'examen du tableau permet de constater que les valeurs liées aux touches vont exclusivement de ØØ à 14; parmi celles-ci, les valeurs attribuées aux touches Ø à F s'entendent d'elles-mêmes. On peut en dire autant des touches de fonctions. Par conséquent, il est tout à fait normal qu'une pression exercée sur l'une des touches fasse apparaître le nombre hexadécimal correspondant sur les afficheurs.

Avant de concevoir un programme, intéressons-nous au moniteur qui comporte le sous-programme SCANDS, lequel fait apparaître sur les afficheurs le contenu des amplacements mémoire ØØF9, ØØFA et ØØFB, ainsi que le montre la figure 14.

Ainsi que nous le savons, les afficheurs à sept segments doivent pouvoir afficher les nombres de Ø à F, ce qui représente seize combinaisons différentes déterminées à l'aide de quatre bits, soit un demi-octet. Quel est exactement le rôle du sous-programme SCANDS? Voici la suite d'opérations qu'il déclenche:

1. Les 4 bits de poids fort de ØØFB sont traduits dans le code à sept segments et ce code est affiché par l'afficheur Di1 pendant une période de 500 μs environ.

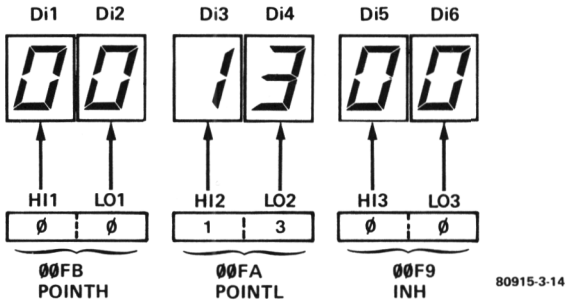


Figure 14. Les afficheurs Di1 . . . Di6 et les tampons de données correspondants.

2. L'afficheur Di1 est déconnecté et les 4 bits de poids faible de ØØFB sont traduits dans le code à sept segments qui est affiché par Di1 pendant 500 μs environ.
3. L'afficheur Di2 est déconnecté et la même opération se déroule pour les contenus des adresses ØØFA et ØØF9 qui sont affichés sur les afficheurs Di3 à Di6.
4. La totalité de l'affichage est déconnectée.

Le sous-programme SCANDS (SCAN DisplayS = lecture de l'affichage) est suivi du sous-programme AK (Accu Key) qui est lancé par une instruction JSR et sert à déterminer si une touche est pressée. Si c'est le cas, le contenu de l'accumulateur est non nul après retour au programme principal. Si aucune touche n'a été pressée, le contenu de l'accumulateur est nul. Ainsi, il est facile de réaliser un branchement déterminé après achèvement du sous-programme en se servant de l'une des instructions BEQ ou BNE.
Dès qu'une touche est pressée, la valeur hexadécimale correspondante doit être traduite dans le code à sept segments adéquat. C'est ce à quoi sert le sous-programme GETKEY du moniteur: Après retour au programme principal, le nombre hexadécimal est stocké dans l'accumulateur du microprocesseur et peut être adressé par l'utilisateur.
La figure 15 nous montre l'ordinogramme simplifié de ces sous-programmes, qui nous aidera à comprendre les conséquences du déroulement de ceux-ci. Ainsi, par exemple, nous allons voir comment la valeur hexadécimale correspondant à une touche est affichée par les afficheurs du Junior Computer. Cette valeur doit apparaître sur les deux afficheurs les plus au centre, tandis que les deux afficheurs respectivement les plus à gauche et les plus à droite afficheront constamment ØØ. C'est la raison pour laquelle le nombre ØØ doit être chargé aux emplacements mémoire ØØFB et ØØF9. D'abord, il est fait appel au sous-programme SCAN1 qui comporte un sous-programme SCANDS + AK grâce auquel le contenu des emplacements mémoire ØØF9, ØØFA et ØØFB est affiché et qui permet de voir si l'une des touches a été pressée. L'instruction de branchement conditionnel BNE maintient le déroulement de la boucle de programme SCAN1 tant qu'une touche n'a pas été enfoncée. Dès que le contenu de l'accumulateur n'est plus nul *et* qu'une touche est relâchée après

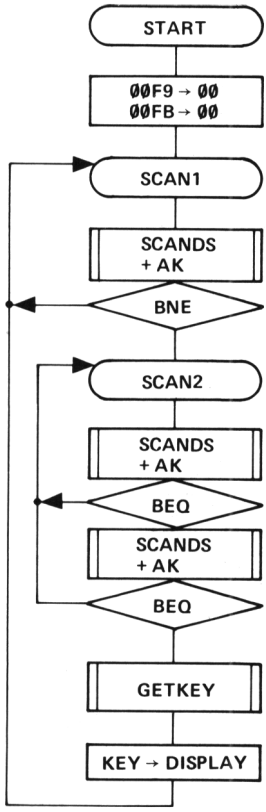


Figure 15. Ordinogramme de la routine moniteur servant à l'interrogation et à l'identification des touches ainsi qu'à l'affichage de la valeur des touches.

avoir été pressée, le microprocesseur quitte la boucle de programme BNE-SCAN1 et parcourt la boucle SCAN2. Désormais, il ne s'agit plus de contrôler qu'une touche a été relâchée. Le sous-programme SCANDS-AK se déroule à nouveau et la boucle BEQ-SCAN2 est parcourue tant que le contenu de l'accumulateur est nul.

Au premier abord, il ne semble pas très raisonnable de faire appel deux fois consécutivement au sous-programme SCANDS (une touche a-t-elle été pressée?) alors que la réponse est connue après que la boucle SCAN1 ait été parcourue. Cependant, ce processus est plus justifié qu'il n'y paraît. C'est tout simplement parce que le microprocesseur met un certain temps à parcourir les boucles de programme SCANDS + AK + BEQ, temps pendant lequel l'effet de rebondissement de la touche pressée disparaît. Ainsi, lorsque le microprocesseur exécute la seconde instruction BEQ, le rebond de la touche a cessé, celle-ci est toujours pressée, et le Junior Computer peut quitter la boucle SCAN2 et procéder au décodage de la valeur de la touche. Si le microprocesseur ne comportait pas ce logiciel

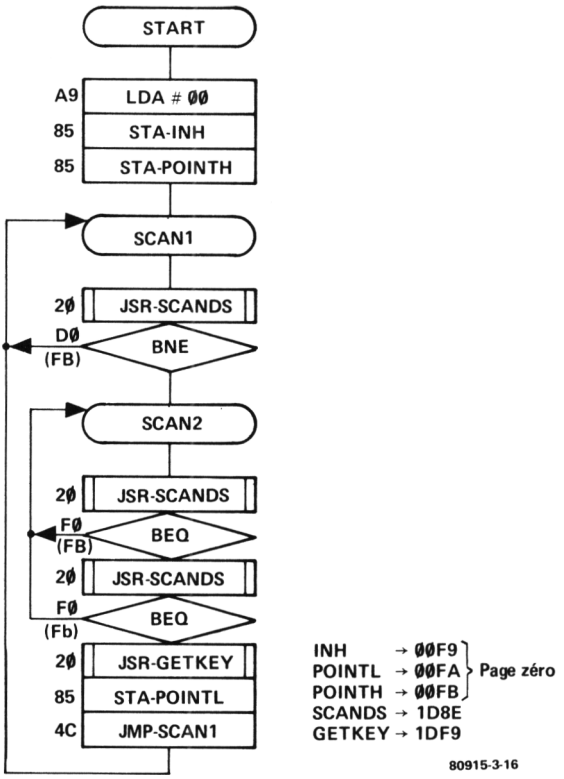


Figure 16. Ordinogramme détaillé du programme de la figure 15.

permettant la réjection du rebond (''software debouncing''), il conclurait faussement qu'une touche, non pressée, l'a été.
A supposer qu'à l'instant où le microprocesseur interroge le clavier, une touche soit pressée, le Junior Computer décode le nombre correspondant à la touche grâce au sous-programme GETKEY et le transfère dans l'accumulateur, puis le copie à l'emplacement mémoire dont l'adresse est ØØFA. Enfin, il revient à la boucle SCAN1, grâce à quoi le nombre apparaît sur les afficheurs. Le micro-ordinateur n'a plus alors qu'à attendre qu'une nouvelle touche soit pressée.
La figure 16 présente l'organigramme détaillé, l'adresse de début étant Ø2ØØ. Les différences entre les deux organigrammes ne sont pas très importantes. Les adresses de début des sous-programmes du moniteur apparaissent clairement. Les instructions RTS et NMI n'y figurent pas, car les touches qui leur correspondent n'ont pas de rapport avec les sous-programmes GETKEY et SCANDS. Le moniteur veille à ce que, après l'exécution d'un sous-programme, l'adresse de l'emplacement mémoire suivant à traiter apparaisse; c'est celui situé à trois adresses plus haut que celle où est stocké le code opération de JSR. Venons-en à

l'utilisation du clavier. N'oublions pas de calculer et de noter les trois déplacements avant de passer à la frappe du programme:

Touches pressées				Affichage		
AD				xxxx	XX	
1	F	D	5	1FD5	D8	
GO				0000	00	
0	B	0	8	0B08	FB	→ noter le déplacement!
1	0	0	D	100D	FB	→ noter le déplacement!
1	5	0	D	150D	F6	→ noter le déplacement!
RST						
AD						
0	2	0	0	0200	XX	
DA		A	9	0200	A9	LDA #
+		0	0	0201	00	
+		8	5	0202	85	STAZ
+		F	9	0203	F9	INH
+		8	5	0204	85	STAZ
+		F	A	0205	FA	POINTL
+		8	5	0206	85	STAZ
+		F	B	0207	FB	POINTH
+		2	0	0208	20	JSR–
+		8	E	0209	8E	SCANDS + AK
+		1	D	020A	1D	
+		D	0	020B	D0	BNE
+		F	B	020C	FB	
+		2	0	020D	20	JSR–
+		8	E	020E	8E	SCANDS + AK
+		1	D	020F	1D	
+		F	0	0210	F0	BEQ
+		F	B	0211	FB	
+		2	0	0212	20	JSR–
+		8	E	0213	8E	SCANDS + AK
+		1	D	0214	1D	
+		F	0	0215	F0	BEQ
+		F	6	0216	F6	
+		2	0	0217	20	JSR–
+		F	9	0218	F9	GETKEY
+		1	D	0219	1D	
+		8	5	021A	85	STAZ
+		F	A	021B	FA	POINTL
+		4	C	021C	4C	JMP–

+		Ø	8	Ø21D	Ø8	SCAN 1
+		Ø	2	Ø21E	Ø2	
AD				Ø21E	Ø2	
Ø	2	Ø	Ø	Ø2ØØ	A9	adresse de début du programme qui se déroulera jusqu'à ce que plus aucune touche ne soit pressée.
GO				ØØØØ	ØØ	
GO				ØØ13	ØØ	valeur de la touche GO
6				ØØØ6	ØØ	valeur de la touche 6
AD				ØØ1Ø	ØØ	valeur de la touche AD
DA				ØØ11	ØØ	valeur de la touche DA
B				ØØØB	ØØ	valeur de la touche B

et ainsi de suite, jusqu'à ce qu'il soit décidé d'arrêter:

RST

Nous constatons qu'après avoir tapé l'adresse de début et, dans l'exemple ci-dessus, avoir pressé deux fois consécutivement la touche GO, le programme se déroule et affiche à chaque pression la valeur correspondant à la touche pressée. Il faut bien noter que la première pression sur GO lance le programme, et que la seconde fait apparaître sur les afficheurs la valeur de GO, laquelle est 13.

Adressage implicite

Dans le jeu d'instructions du 6502, vingt-cinq d'entre elles concernent des tâches internes. Il s'agitt, par exemple, du transfert interne de données d'un registre à un autre. Certaines agissent sur le déroulement du programme. L'opérande d'une instruction implicite est contenue dans le code opération. Par conséquent, nous traitons des instructions d'un octet. Nous connaissons déjà certaines d'entre elles que nous retrouverons dans la liste ci-dessous:

BRK code ØØ BReaK (Interruption software) (ou simulée)
CLC code 18 CLear Carry (Annuler retenue)
CLD code D8 CLear Decimal mode (Annuler le mode décimal)
CLI code 58 CLear Interrupt flag (Annuler inhibition des interruptions)
CLV code B8 CLear oVerflow flag (Annuler débordement)
DEX code CA DEcrement X-register by one (Décrémenter X)
DEY code 88 DEcrement Y-register by one (Décrémenter Y)
INX code E8 INcrement X-register by one (Incrémenter X)
INY code C8 INcrement Y-register by one (Incrémenter Y)
NOP code EA No OPération (pad d'opération) instruction muette
PHA code 48 PusH Accumulator on stack (Empiler A)
PHP code Ø8 PusH Processor-status on stack (Empiler P)
PLA code 68 PulL Accumulator from stack (Dépiler A)
PLP code 28 PulL Processor-status from stack (Dépiler P)
RTI code 4Ø ReTurn from Interrupt (Retour d'interruption)
RTS code 6Ø ReTurn from Subroutine (Retour de sous-programme)

SEC code 38 SEt Carry flag (Mettre retenue à 1)
SED code F8 SEt Decimal flag (Mettre en mode décimal)
SEI code 78 SEt Interrupt flag (Inhiber les interruptions)
TAX code AA Transfer Accumulator to X (Transférer A dans X)
TAY code A8 Transfer Accumulator to Y (Transférer A dans Y)
TSX code BA Transfer Stackpointer to X (Transférer S dans X)
TXA code 8A Transfer X to Accumulator (Transférer X dans A)
TXS code 9A Transfer Y to Stack pointer (Transférer X dans S)
TYA code 98 Transfer Y to Accumulator (Transférer Y dans A)
Examinons-en quelques-unes plus en détail.

SED et CLD

Les instructions ADC et SBC permettent d'ajouter, ou de soustraire du contenu de l'accumulateur, le contenu d'une adresse-mémoire. En ce qui concerne l'addition de deux nombres, on peut dire qu'il y a retenue lorsque le résultat est supérieur, par exemple à 11111111 ou FF. Le chapitre 2 nous l'a appris. En tout état de cause, le flag Carry est mis à 1. A partir de ce point, il est possible d'effectuer dans l'accumulateur des calculs "binaires", ainsi que des calculs décimaux. L'exécution d'une instruction SED met à 1 le bit D du registre d'état le microprocesseur se trouve en mode décimal pour les instructions ADC et SBC. Une instruction CLD met à Ø l'indicateur (flag) D, grâce à quoi le mode binaire (hexadécimal) opérera pour les instructions ADC et SBC.
Rappelons-nous que dans l'addition hexadécimale, lorsque le contenu de l'accumulateur dépasse 11111111 (ou FF), et que dans l'addition décimale il est supérieur à 1ØØ11ØØ1 (ou 99, car le code BCD de 9 est 1ØØ1), le flag Carry est mis à 1.
Lorsqu'on additionne en mode binaire, il faut toujours commencer par une instruction CLD, car le fonctionnement de l'ordinateur est sensiblement différent selon qu'il calcule dans un mode ou dans l'autre. L'instruction SED a pour effet de permettre le retour au mode décimal; l'on termine par une instruction CLD.
Notons également qu'après l'initialisation du programme moniteur (par une pression sur la touche RST), le Junior Computer revient automatiquement au mode binaire (D = Ø).

NOP (No OPération)

Ainsi que le nom l'indique, cette instruction n'a aucune influence sur le déroulement d'un programme. Alors à quoi peut-elle bien servir? Supposons que nous ayons écrit un programme comportant quelques centaines de lignes (emplacements mémoire) et que nous voulions ultérieurement y introduire des instructions complémentaires. Si, malheureusement, nous n'avons pas ménagé d'emplacements sans aucune opération, il ne reste plus qu'à réécrire entièrement le programme. C'est d'autant plus pénible qu'il ne s'agit parfois que de rajouter un ou deux octets. Par conséquent, à moins d'avoir l'absolue certitude qu'un tel cas ne peut se produire pour un programme déterminé, la sagesse commande de réserver quelques emplacements en incluant aux points stratégiques du programme quelques instructions NOP. Par la suite, elles seront surchargées par les

opérandes d'autres codes opération. Elles peuvent servir également à combler les vides laissés par d'éventuelles corrections de programme.

Instructions de transfert

Toutes les instructions implicites dont les mnémoniques commencent par les lettres P ou T se rapportent à des transferts internes de données en direction ou en provenance de A, X, Y, S ou P, ou de la pile, sur la page Ø1.

Il est fréquent qu'au cours du déroulement d'un programme, le contenu d'un registre interne doive être sauvegardé. Si, par exemple, nous sautons du programme principal à un sous-programme, le registre X est indispensable à chacun d'eux. Il est donc évident qu'il convient dans ce cas de sauvegarder le contenu du registre X dans un endroit quelconque de la mémoire avant de procéder au branchement vers le sous-programme. Après exécution de celui-ci, il conviendra d'aller rechercher l'ancien contenu et de le restaurer afin de pouvoir poursuivre le déroulement du programme principal:

TXA	transférer le contenu de X dans l'accumulateur
PHA	déposer le contenu de l'accumulateur sur la pile
JSR-XXXX	sauter à un sous-programme (et utiliser X selon les nécessités)
RTS	retour au programme principal
PLA	transférer le contenu de la pile dans l'accumulateur
TAX	transférer le contenu de l'accumulateur dans X

Mais, il existe une autre procédure:

STX-SAVX	sauvegarder le contenu de X à l'emplacement mémoire SAVX
JSR-XXXX	sauter au sous-programme (le registre X est disponible)
RTS	retour au programme principal
LDX-SAVX	transférer le contenu de l'emplacement mémoire SAVX dans le registre X

Les instructions STX et LDX requièrent davantage d'octets. La première procédure est donc plus intéressante et le registre S constitue une mémoire temporaire auxiliaire idéale.

Il n'est pas rare que, avant d'exécuter un saut à un sous-programme, il soit nécessaire de sauvegarder le contenu de tous les registres du microprocesseur (dans la pile) afin de pouvoir continuer à les exploiter après le retour au programme principal. A cette fin, on se sert, avant le saut, du programme de sauvegarde SAVE et d'un programme RESTO, restaurant leur état original. La figure 17 présente les deux programmes, appelés également routines. SAVE permet de transférer le contenu de A, puis ceux des registres X, Y et P sur la pile, l'accumulateur servant de station intermédiaire de stockage. En ce qui concerne RESTO, il en va inversement. La pile est dépilée, contenu après contenu, chacun de ceux-ci étant transféré dans le registre concerné, par l'intermédiaire de l'accumulateur. Nous constatons que le premier contenu dépilé est celui qui avait été le dernier empilé. C'est le résultat du fonctionnement du pointeur de pile (stack pointer) dont nous vous avons déjà parlé dans notre exposé sur les sous-programmes.

Le pointeur de pile (fixé par l'intermédiaire de S) indique continuellement le prochain emplacement libre de la pile. Après l'initialisation du moniteur du Junior Computer (résultant d'une pression sur RST), le pointeur est

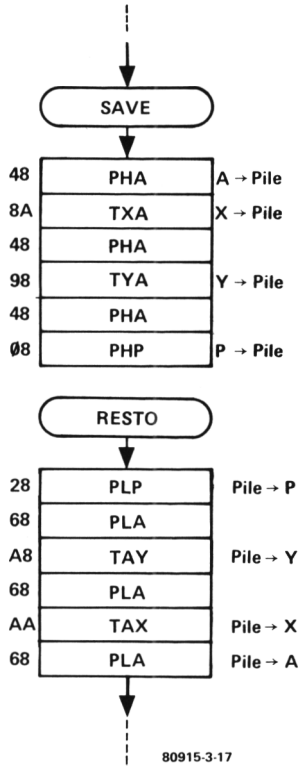


Figure 17. Sauvegarde du contenu des registres internes du microprocesseur 6502 sur la pile, suivie de la restauration, à l'aide d'instructions d'empilage et de dépilage.

automatiquement dirigé vers l'adresse Ø1FF de la page 1 de la RAM. Mais, il est possible d'orienter le pointeur vers une autre adresse de la page 1, et voici comment:

```
LDX #81    charger 81 dans le registre X
TXS        transférer X dans le registre S
```

Dès cet instant, le pointeur indique l'adresse Ø181 de la page 1. L'adresse Ø1FF est celle du dernier emplacement de la pile, et comme le pointeur est dirigé vers une autre direction, il existe une forte chance pour que le nombre maximal d'emplacements de la pile soit dépassé. Cette situation peut être signalée par la routine STKCHK, ainsi que le montre la figure 18. Dès que le pointeur indique Ø1ØF, un signal d'erreur est généré sous la forme de l'apparition de EEEEEE sur l'affichage. A ce moment, il reste 16 emplacements libres dans la pile. Ce n'est qu'après avoir actionné la touche RST que le retour au programme moniteur s'effectue. Attention: si le pointeur de pile se trouve sur Ø1ØØ et qu'on tente d'introduire de nouvelles données dans la pile, le pointeur redémarre à partir de Ø1FF.

Adressage de l'accumulateur

C'est une variante de l'adressage implicite, regroupant quatre instructions

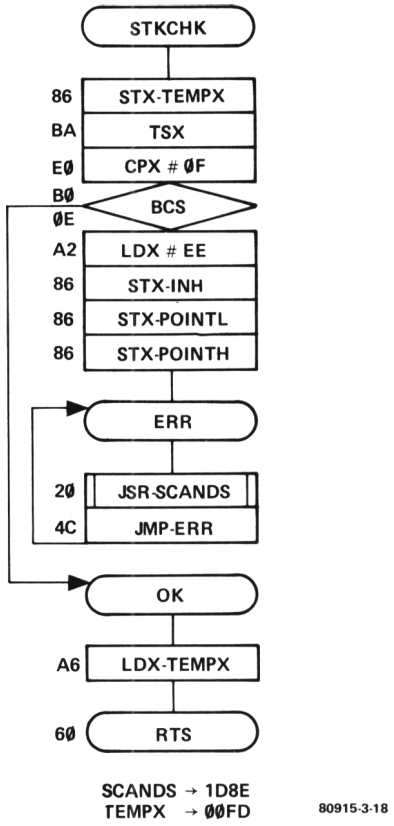


Figure 18. Ordinogramme du programme STKCHK grâce auquel le dépassement de la capacité de la pile est signalé par un message d'erreur (EEEEEE) apparaissant sur les afficheurs.

d'un octet et permettant de manipuler le contenu de l'accumulateur. Elles concernent le décalage (shift) grâce à ASL et LSR, et la rotation (rotate) à l'aide de ROL et ROR. Elles s'utilisent pour des opérations arithmétiques.

LSR (Logical Shift Right). Décalage logique à droite; code opération 4A. Les bits sont décalés vers la droite:

b7	b6	b5	b4	b3	b2	b1	b0	
P	Q	R	S	T	U	V	W	
		↓						
		LSR						
		↓						
0	P	Q	R	S	T	U	V	Carry flag = W

La rangée supérieure indique la position des bits. Les lettres P à W représentent 1 ou Ø. Nous constatons que tous les bits sont décalés d'un emplacement vers la droite, le bit le plus à gauche devient Ø, le bit bØ = W détermine le flag Carry. Si W est un Ø, le flag Carry est mis à "Ø"; si W est un 1, le flag Carry est mis à "1". Le flag N reste à "Ø". Huit décalages de suite ont pour résultat que le contenu de l'accumulateur est ØØØØØØØØ, seul le flag Z est mis à "1".

ASL (Arithmetic Shift Left). Décalage arithmétique à gauche, code opération ØA. Les bits sont décalés vers la gauche:

b7	b6	b5	b4	b3	b2	b1	bØ	
P	Q	R	S	T	U	V	W	
		↓						
		ASL						
		↓						
P	Q	R	S	T	U	V	W	Ø Carry flag = P

Tous les bits sont décalés d'un emplacement vers la gauche; le bit le plus à droite devient Ø et le bit b7 détermine le flag Carry, P, qui est mis à "1" lorsque P est 1, ou à Ø lorsque P est Ø. Le flag N est mis à "1" dès que le bit sept, Q, est 1. Quand le contenu de l'accumulateur est ØØØØØØØØ, le flag Z est mis à "1", ce qui se produit après huit instructions ASL consécutives.

ROL (ROtate Left). Rotation à gauche; code opération 2A.

b7	b6	b5	b4	b3	b2	b1	bØ	
P	Q	R	S	T	U	V	W	C
		↓						
		ROL						
		↓						
Q	R	S	T	U	V	W	$C_{antérieur}$	$C_{nouveau}$ = P

Dans ce cas également, les bits sont décalés d'un emplacement vers la gauche. La différence avec l'instruction ASL réside dans le fait que seul le bit bØ devient égal à la valeur antérieure de C. L'état des autres flags est identique à celui indiqué pour l'instruction ASL.

ROR (ROtate Right). Rotation à droite; code opération 6A.

b7	b6	b5	b4	b3	b2	b1	bØ	
P	Q	R	S	T	U	V	W	C
		↓						
		ROR						
		↓						
C_{ant}	P	Q	R	S	T	U	V	$C_{nouveau}$ = W

Tous les bits sont décalés d'un emplacement vers la droite. Par rapport à l'instruction LSR, la différence réside dans le fait que le septième bit

n'est pas devenu Ø, mais prend la valeur antérieure de C. L'état des autres flags est identique à celui indiqué pour l'instruction LSR.
En fait, la différence entre le décalage et la rotation résulte de ce que, à la suite d'un décalage, un bit du contenu de l'accumulateur est perdu, alors qu'avec la rotation ce n'est pas le cas.
Les quatre instructions du décalage et de la rotation que nous venons d'étudier peuvent être aussi réalisées par la combinaison du flag Carry et du contenu d'un emplacement mémoire. Il s'agit alors d'instructions d'adressage absolu ou de variantes de celles-ci.

Décalage et rotation

Nous savons que lorsque nous inscrivons des données en les tapant sur le clavier du Junior Computer, elles se déplacent de la droite vers la gauche sur les afficheurs, au fur et à mesure de leur introduction. Nous savons également, à l'examen de la figure 14, que les données à afficher sont stockées dans les trois tampons (buffers) de données, les emplacements mémoire dont les adresses sont ØØFB, ØØFA et ØØF9. La figure 19 illustre le déplacement des données. Dès qu'une nouvelle touche est pressée, la routine SHIFT du programme moniteur démarre, ce qui a pour conséquence que tous les bits des trois tampons sont décalés de quatre emplacements vers la gauche. Quatre bits sont précisément suffisants à l'alimentation d'un caractère d'affichage, et, après l'instruction SHIFT les cinq caractères les plus à gauche sur l'affichage sont décalés d'un emplacement vers la gauche; celui qui occupait la position la plus à gauche desparaît et celui qui apparaît le plus à droite correspond à la touche qui vient d'être pressée et a provoqué l'appel de la routine SHIFT.
Pendant que les quatre bits successifs sont décalés, le flag Carry sert de relais. D'abord, le bit b7 de INH est décalé dans le flag Carry à la suite d'une instruction ASLZ (l'adjonction de Z à ASL signifie que la page d'adressage zéro est utilisée, code opération Ø6) et devient le bit nul Ø. Ensuite, le programme moniteur déclenche une instruction ROLZ (code opération 26) qui provoque une rotation du tampon POINTL; le flag Carry est décalé dans le bit bØ de POINTL, ce qui revient à dire que c'est le bit b7 de INH qui se trouve décalé dans le bit bØ de POINTL, par l'intermédiaire du flag Carry. Simultanément, le bit b7 de POINTL est décalé dans le flag Carry.
Suit une nouvelle instruction ROLZ, lancée par le moniteur. De ce fait, le flag Carry est décalé dans le bit bØ de POINTH. Mais, ce flag Carry est le bit b7 de POINTL, ce qui veut dire que ce bit b7 de POINTL est décalé dans le bit bØ de POINTH, par l'intermédiaire du flag Carry. Simultanément, le bit b7 de POINTH est décalé dans le flag Carry.
Le même processus se répète quatre fois de suite. Finalement, les quatre bits initiaux les plus à gauche de POINTH ont disparu; les quatre bits les plus à droite de INH sont devenus des Ø. Les deux derniers rectangles de la figure 19 concernent l'utilisation de ces quatre bits à l'affichage de la valeur de la touche pressée.
Maintenant, nous allons aborder un programme dans lequel nous utiliserons un certain nombre des instructions et des routines du programme moniteur mentionnées ci-dessus.

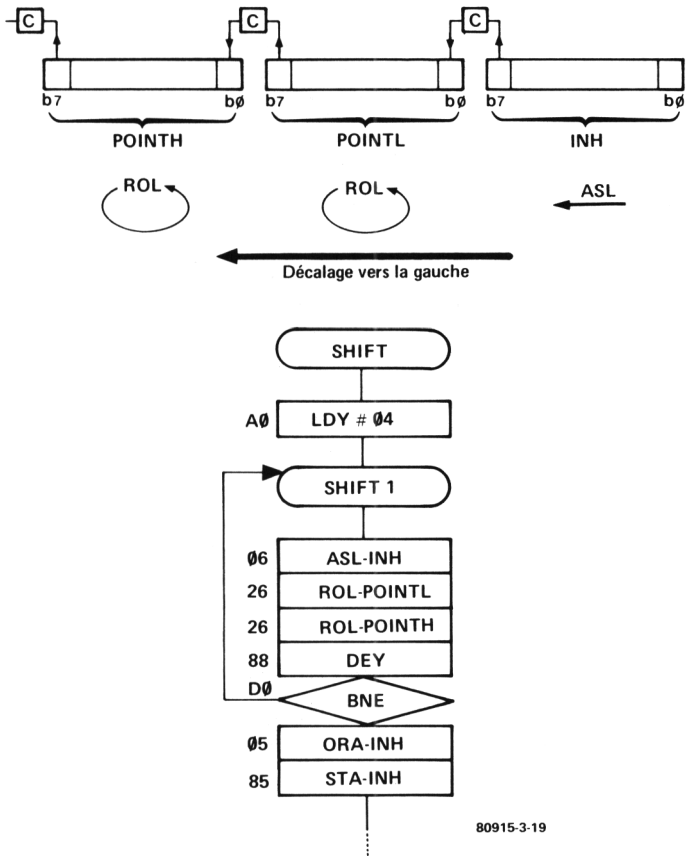


Figure 19. Programme grâce auquel les données correspondant aux touches frappées se décalent de la droite vers la gauche sur les afficheurs. Cette routine fait aussi usage d'une partie du programme moniteur.

Addition décimale

Il est certain que pour effectuer l'addition décimale de deux nombres de six chiffres, une calculatrice de poche, ou, à la rigueur, un crayon et une feuille de papier conviendraient mieux que le Junior Computer. Mais, là n'est pas la question. L'exécution d'une addition grâce à notre micro-ordinateur nous permettra d'apprendre à l'utiliser à des tâches sensiblement plus complexes, et plus importantes.
Soit, l'addition suivante:

X X X X X X	+	X X X X X X	=	X X X X X X	
premier nombre		second nombre		résultat	

Les chiffres apparaîtront sur les afficheurs et s'y déplaceront de la droite

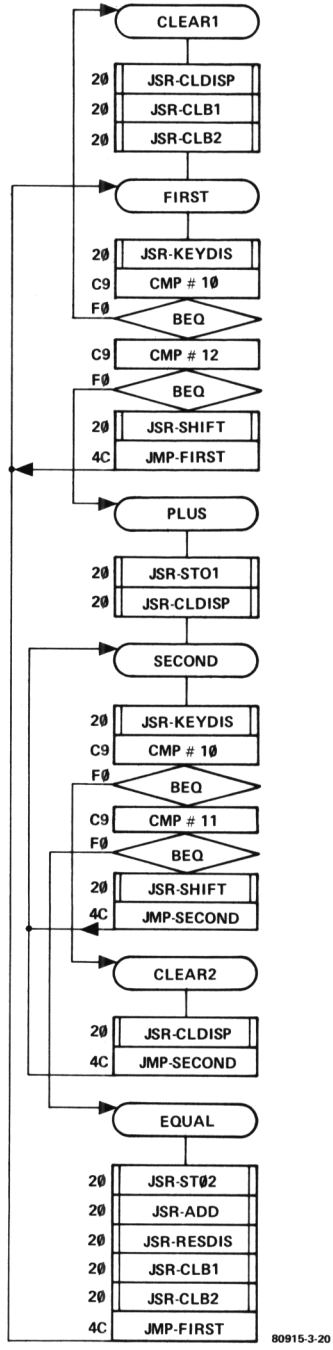
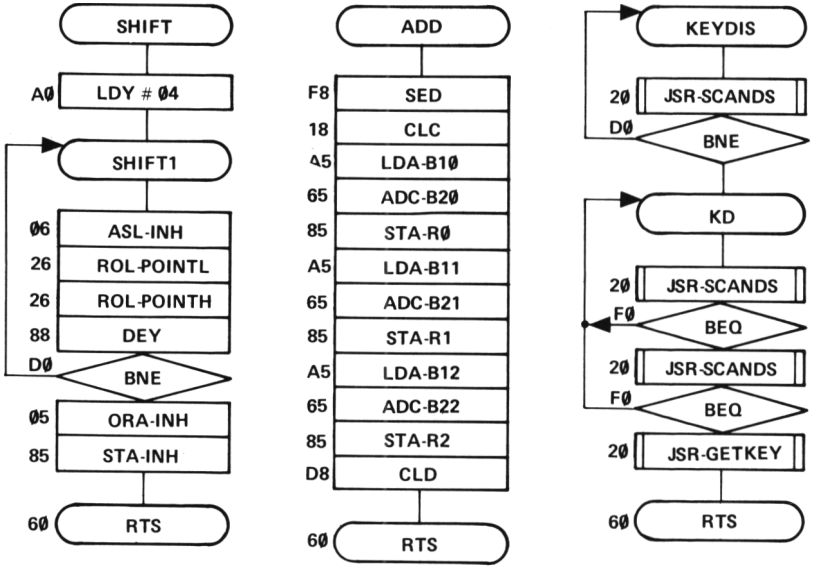


Figure 20. Programme principal de l'addition décimale de deux nombres de six chiffres.

vers la gauche et seront donc tapés, grâce aux touches du clavier, dans l'ordre où on les inscrirait sur une feuille de papier, c'est-à-dire, le chiffre des centaines de mille en premier lieu, puis celui des dizaines de mille, etc... Avant d'introduire les nombres, il faut que l'extinction des afficheurs soit réalisée automatiquement. Les touches Ø à 9 serviront à l'écriture des chiffres. La touche DA, dont la valeur est 11, jouera le rôle du symbole = et, avec la touche AD, dont la valeur est 10, l'exinction des afficheurs sera commandée. L'utilisation de la touche + s'impose. Si le résultat de l'addition est supérieur à 999999, il n'est pas nécessaire qu'un message d'erreur soit émis; nous traiterons de ce sujet d'autre part. A supposer qu'une erreur soit commise lors de l'écriture d'un nombre, celui-ci doit pouvoir être annulé grâce à la touche d'effacement, après quoi le nombre exact est introduit.
La figure 20 montre l'ordinogramme de l'addition décimale de deux nombres. Un examen attentif du programme principal, débutant par CLEAR1, révèle qu'il se compose de neuf sous-programmes se succédant et remplissant chacun une tâche précise. La figure 21 en présente le détail.
Il existe 4 x 3 = 12 emplacements mémoire (tampons) dont les adresses sont les suivantes:

POINTH	POINTL	INH	Tampons (buffers) pour l'affichage
ØØFB	ØØFA	ØØF9	
B12	B11	B1Ø	Tampons pour le premier nombre
ØØØ2	ØØØ1	ØØØØ	
B22	B21	B2Ø	Tampons pour le second nombre
ØØØ5	ØØØ4	ØØØ3	
R2	R1	RØ	Tampons pour le résultat de l'addition
ØØØ8	ØØØ7	ØØØ6	

Chaque tampon est affecté à deux afficheurs. L'ordre de gauche à droite est identique à l'ordre d'inscription normal d'un nombre.
Reportons-nous au programme de la figure 20 et commençons par CLEAR1 qui a pour effet de mettre à ØØØØØØØØ le contenu des tampons d'affichage et des nombres (voir aussi les figures 21d, 21e et 21f). Ensuite vient FIRST, le programme saute au sous-programme KEYDIS (figure 21c) qui applique les sous-programmes SCANDS et GETKEY à la commande des afficheurs. Par l'intermédiaire de ces deux sous-programmes, KEYDIS interroge les touches et neutralise leur rebond. Si l'une des touches est pressée, GETKEY la détermine. Après le retour de KEYDIS au programme principal, la valeur de la touche pressée est présente dans l'accumulateur.
Ensuite, c'est le programme principal qui permet de savoir si l'une des touches CLEAR (autrement dit, AD) ou + a été pressée, grâce respectivement à l'instruction CMP # 10 et à l'instruction CMP # 12.



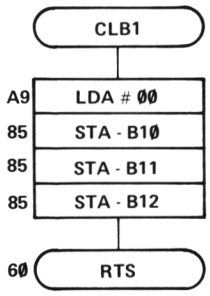
Figures 21a à 21i. Sept sous-programms associés au programme de la figure 20.

Si ce n'est pas le cas, ce ne peut être que l'une des touches de Ø à 9, ou la touche = (soit, DA). Avec le sous-programme SHIFT, qui fait suite, le chiffre est décalé dans le tampon d'affichage, puis affiché par le sous-programme KEYDIS.

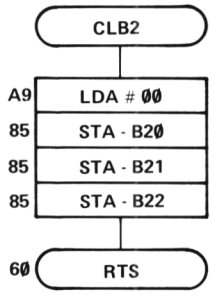
Lorsque CLEAR (ou AD) est pressée, le programme saute à CLEAR1. Si, au contraire, c'est la touche +, le programme se branche sur PLUS d'où un saut sera effectué au sous-programme STO1 (figure 21h). STO1 transfère le nombre, qui vient d'être introduit, des tampons d'affichage aux tampons du premier nombre. Par le fait même, les tampons d'affichage peuvent accueillir le second nombre, après que, par l'intermédiaire du sous-programme CLDISP (figure 21f), ils aient été remis à zéro.

La prise en compte du second nombre intervient grâce à la partie du programme baptisée SECOND. Le programme principal se branche d'abord sur le sous-programme KEYDIS qui commande l'affichage et le clavier comme pour le premier nombre. Après qu'une touche ait été pressée, KEYDIS revient au programme principal et la valeur de la touche est chargée dans l'accumulateur. Puis, s'il s'agissait d'une touche numérique, Ø à 9, le second nombre est décalé dans les tampons d'affichage. Ensuite, le programme principal détecte si l'une des touches CLEAR (ou AD) ou + a été pressée. A supposer que ce soit CLEAR, le dernier nombre inscrit est effacé, grâce au sous-programme CLDISP.

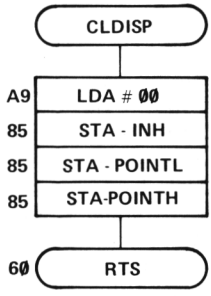
Mais, il se peut aussi que ce soit la touche = (ou DA), ce qui signifie alors, pour le programme principal, que l'introduction des nombres est terminée et que l'addition de ceux-ci peut être effectuée. Débute donc la



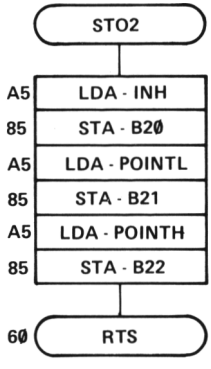
80915-3-21d



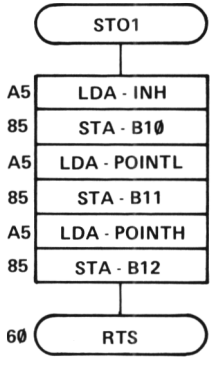
80915-3-21e



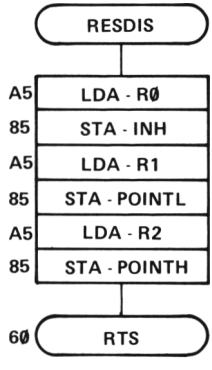
80915-3-21f



80915-3-21g



80915-3-21h



80915-3-21i

B1Ø → ØØØØ
B11 → ØØØ1
B12 → ØØØ2

B2Ø → ØØØ3
B21 → ØØØ4
B22 → ØØØ5

RØ → ØØØ6
R1 → ØØØ7
R2 → ØØØ8

INH → ØØF9
POINTL → ØØFA
POINTH → ØØFB

SCANDS → 1D8E
GETKEY → 1DF9

partie du programme baptisée EQUAL. D'abord, le sous-programme STO2 transfère le dernier nombre inscrit des tampons d'affichage dans les tampons des nombres (figure 21g), et les afficheurs libérés sont en mesure d'afficher le résultat. Le sous-programme ADD (voir figure 21b) effectue l'addition décimale des deux nombres et le résultat est déposé dans les tampons R2-R1-RØ. Le sous-programme RESDIS (figure 21i) transfère le contenu des tampons de résultat dans les tampons d'affichage. Les sous-programmes CLB1 et CLB2 remettent à Ø les tampons des nombres et le Junior Computer est prêt à l'inscription de deux nouveaux nombres. Le sous-programme KEYDIS provoque l'affichage du résultat.
Ce programme d'addition permet de constater que l'emploi des sous-programmes rend plus accessible la compréhension du programme principal.

Adressage indexé absolu, adressage X

Les instructions dont l'adressage est absolu comportent trois octets. Le premier est le code opération, les deux autres (ADL et ADH) précisent l'adresse de l'opérande.

L'adressage absolu indexé X est une variante du précédent. Il comporte aussi trois octets, le premier pour le code opération, le second pour l'ADL et le troisième pour l'ADH d'une adresse, qui, en général, n'est pas celle de l'opérande. L'opérande est stockée à une adresse résultant de l'addition du contenu du registre index X à la partie adresse de l'instruction.

Avant que nous ne passions à l'étude plus détaillée de ce mode d'adressage, il faut que nous abordions la notion de "champ" (Field). Par champ, on entend un assemblage de données stockées l'une derrière l'autre dans une section quelconque de la mémoire du microprocesseur. L'index X est extrêmement pratique pour traiter plusieurs champs, dont un certain nombre sont des champs d'opérandes et d'autres des champs de résultats. Les opérations se rapportent à des positions concordantes dans chacun des champs d'opérandes et le résultat de ces opérations apparaît sur des positions concordantes des champs de résultats.

Voici un exemple regroupant les champs FIELD 1, 2 et 3:

FIELD1:	Ø12Ø	FIELD2:	ØØ11	FIELD3:	Ø3ØØ
Ø12Ø	Ø1	ØØ11	1Ø	Ø3ØØ	11
Ø121	Ø2	ØØ12	2Ø	Ø3Ø1	22
Ø122	Ø3	ØØ13	3Ø	Ø3Ø2	33
Ø123	Ø4	ØØ14	4Ø	Ø3Ø3	44
Ø124	Ø5	ØØ15	5Ø	Ø3Ø4	55
Ø125	Ø6	ØØ16	6Ø	Ø3Ø5	66
Ø126	Ø7	ØØ17	7Ø	Ø3Ø6	77
Ø127	Ø8	ØØ18	8Ø	Ø3Ø7	88
Ø128	Ø9	ØØ19	9Ø	Ø3Ø8	99
Ø129	ØA	ØØ1A	AØ	Ø3Ø9	AA

Les champs 1 et 2 comportent des nombres additionnés selon l'alignement horizontal, le résultat étant donné sur la même ligne du champ 3. Les champs sont caractérisés par leur adresse de début. C'est le contenu du registre index X qui précise, en un moment donné, la ligne d'addition.

La figure 22 montre le programme d'addition conforme au tableau précédent, avec l'utilisation de l'adressage absolu indexé X. L'addition proprement dite s'opère grâce au sous-programme IND1. L'accumulateur est chargé du contenu d'un emplacement mémoire déterminé du champ FIELD1.

Après une instruction CLC (carry C = Ø), l'instruction ADC permet d'y ajouter le contenu de l'emplacement de mémoire correspondant de FIELD2. Le résultat de l'addition est déposé dans l'emplacement mémoire correspondant de FIELD3 (STA).

Le programme débute par le chargement de Ø9 dans X. Ensuite a lieu l'addition déjà décrite, puis le contenu de X est décrémenté de 1. La première addition concerne donc les adresses Ø12Ø + Ø9 = Ø129 de FIELD1, ØØ11 + Ø9 = ØØ1A de FIELD2 et Ø3ØØ + Ø9 = Ø3Ø9 de FIELD3.

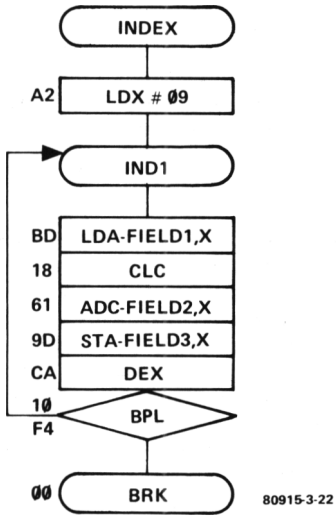


Figure 22. Programme de l'addition d'un nombre 1 d'un champ 1 avec un nombre 2 concordant, du champ 2; le résultat apparaît à la position correspondante du champ 3. C'est l'addition tabulée.

En d'autres termes, on commence à la ligne inférieure du tableau et l'on opère de bas en haut.

Remarquons la notation de l'instruction, par exemple, LDA-FIELD1,X. D'abord vient la définition de l'opération LDA, puis un tiret, qui indique l'adressage absolu, ensuite l'indication du champ, suivie d'une virgule, et enfin un X. Autrement dit, d'abord le code opération, ensuite l'ADL et l'ADH de la première adresse du champ correspondant.

En outre, l'addition peut s'effectuer avec adressage absolu, par exemple:

premier octet:	LDA-	code opération
deuxième octet:	ADL	de la donnée à charger
troisième octet:	ADH	de la donnée à charger
quatrième octet:	CLC	(carry C = Ø)
cinquième octet:	ADC-	code opération
sixième octet:	ADL	de la donnée à additionner avec le contenu de l'accumulateur
septième octet:	ADH	de la donnée à additionner avec le contenu de l'accumulateur
huitième octet:	STA-	code opération
neuvième octet:	ADL	du contenu de l'accumulateur à stocker
dixième octet:	ADH	du contenu de l'accumulateur à stocker

Ainsi est exécutée l'addition d'une ligne du tableau. Pour un tableau identique à celui de notre exemple, comportant des champs de dix emplacements mémoire, l'exécution de la seule opération d'addition mobilise déjà 10 x 10 = 100 emplacements mémoire. Reportons-nous au programme de la figure 22 et constatons qu'en tout et pour tout, il faut 17

emplacements mémoire. C'est là qu'apparaît le grand avantage de l'adressage indexé: il nécessite moins d'emplacements mémoire. En outre, le programme est nettement plus clair.
La figure 23 détaille un peu plus le programme de la figure 22.
Il s'agit d'un instantané du registre index X où est présente la donnée Ø4.
Les instructions de l'adressage absolu indexé X sont réunies à la fin de ce livre.
Il arrive souvent que des *blocs de données* doivent être déplacés d'une section de mémoire à une autre. C'est ce qu'illustre le programme MOVE de la figure 24. C'est le transfert d'un champ (FROMAD, adresse Ø172) à un autre (TOAD, adresse Ø3AØ). Nous constatons que l'accumulateur sert de station relais: LDA-, suivi par STA-. Le bloc comporte cinq données. La donnée Ø4 est d'abord déposée dans le registre X (LDX # Ø4), après quoi son contenu est décrémenté de 1 et le processus se reproduit jusqu'à ce que le contenu de X soit égal à Ø. A ce moment, toutes les données ont été déplacées, ou plus exactement, copiées, (adresse de début Ø172 + Ø4 = Ø176). Dès que le contenu du registre X est négatif, la condition du branchement BPL n'est plus remplie et le programme stoppe. Voici comment sont entrées les données:

Touches pressées				**Affichage**		
AD				xxxx	xx	
Ø	2	Ø	Ø	Ø2ØØ	xx	
DA		A	2	Ø2ØØ	A2	LDX#
+		Ø	4	Ø2Ø1	Ø4	
+		B	D	Ø2Ø2	BD	LDA-,X
+		7	2	Ø2Ø3	72	ADL de FROMAD
+		Ø	1	Ø2Ø4	Ø1	ADH de FROMAD
+		9	D	Ø2Ø5	9D	STA-,X
+		A	Ø	Ø2Ø6	AØ	ADL de TOAD
+		Ø	3	Ø2Ø7	Ø3	ADH de TOAD
+		C	A	Ø2Ø8	CA	DEX
+		1	Ø	Ø2Ø9	1Ø	BPL
+		F	7	Ø2ØA	F7	déplacement
+		4	C	Ø2ØB	4C	JMP
+		3	3	Ø2ØC	33	ADL de l'adresse moniteur
+		1	C	Ø2ØD	1C	ADH de l'adresse moniteur
AD				Ø2ØD	1C	
Ø	2	Ø	Ø	Ø2ØØ	A2	
GO				Ø2ØØ	A2	début de programme

A la fin du programme, le microprocesseur revient au programme moniteur (adresse IC33); après l'exécution du programme, l'adresse de début est affichée.
N.B. Le registre X contient huit octets. On ne pourra par conséquent, procéder au transfert de blocs de données supérieurs à 256 octets.

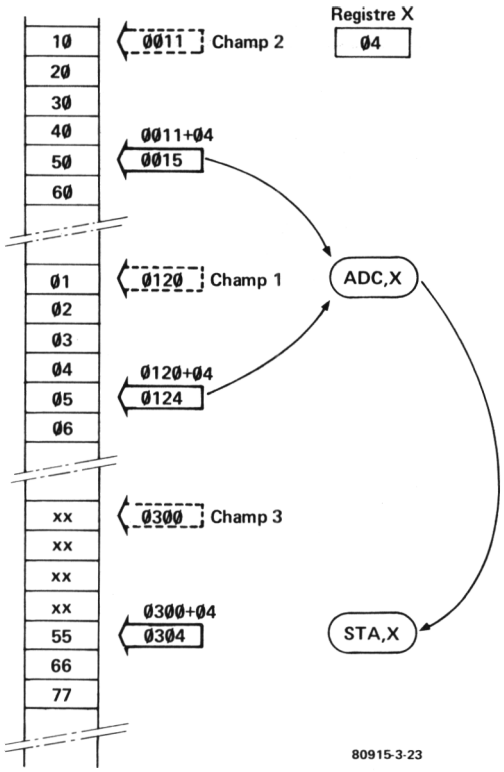


Figure 23. Instantané du programme de la figure 22, dans le cas où X = 04.

Adressage absolu indexé Y

Ce mode d'adressage ne mérite pas d'explications approfondies car c'est une forme particulière de l'adressage absolu indexé qui remplit la même fonction que l'adressage indexé X, la seule différence entre les deux modes étant que, dans le premier cas, le registre X sert de registre index, alors que dans le second cas c'est le registre Y qui joue ce rôle. L'avantage de disposer de deux registres réside dans le fait qu'il est possible d'imbriquer l'une dans l'autre deux boucles de programme. Nous verrons cet aspect plus en détail dans le cours des prochains paragraphes. Mais, attention, toutes les instructions ne peuvent être utilisées dans les deux modes X et Y. Les codes opération sont indiqués dans les annexes en fin de livre.

Adressage page-zéro indexé X

L'adressage page-zéro est une variante de l'adressage absolu et l'adressage page-zéro indexé X est une variante de l'adressage absolu indexé X. Dans les deux cas, seul l'octet d'adresse droit ADL de l'adresse de l'opérande

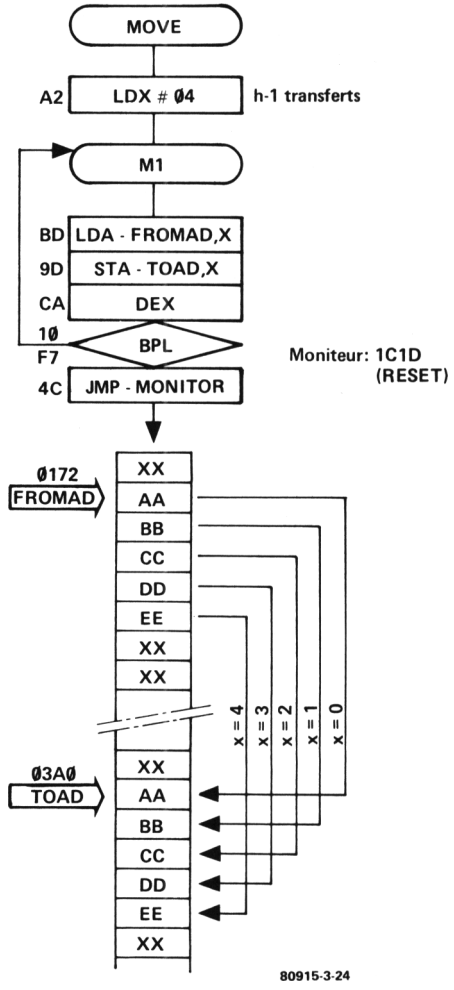


Figure 24. Programme de transfert d'un bloc de données.

doit être précisé; l'octet d'adresse gauche ADH est déterminé: ØØ. L'avantage vient de ce que l'instruction ne comporte que deux octets au lieu de trois. Pour le reste, la procédure d'utilisation est identique à celle de l'adressage absolu. Les codes opération se trouvent à la fin du livre.

Adressage page-zéro indexé Y

Ce mode d'adressage remplit les mêmes fonctions que celui que nous venons de voir, la seule réserve étant que toutes les instructions ne peuvent être utilisées dans les deux modes. Nous venons d'examiner quatre formes

d'adressage indexé, et nous savons que deux registres sont à tout moment utilisables pour ce genre d'adressage. D'ailleurs, jusqu'à présent, ce sont les modes d'adressage direct qui ont retenu toute notre attention, mais nous abordons maintenant l'adressage indirect.

L'adressage indirect

En abordant l'étude de l'adressage indirect, nous ne vous cacherons pas qu'elle présente quelques difficultés. Mais, à l'aide de quelques exemples, nous nous familiariserons rapidement avec ce mode qui nous permettra d'écrire des programmes courts et clairs. Autre avantage, il n'est pas nécessaire que les adresses des emplacements dans lesquels il faudra écrire ou lire des données soient connues au moment de l'élaboration du programme. Le programmeur pourra les stocker plus tard en page-zéro, ou laisser à l'ordinateur le soin de les calculer. Ce mode d'adressage est certainement l'une des raisons du succès du 6502.

Adressage indexé indirect

Avant tout, faisons un léger retour en arrière. La figure 25a nous montre comment, à l'aide de l'adressage indexé Y, le contenu de l'emplacement mémoire Ø21A est transféré dans l'accumulateur. Le contenu du registre Y est ØØ. La partie du programme qui concerne le chargement apparaît comme suit:

```
B9   LDA,Y
1A   ADL de l'octet d'adresse (opérande)
Ø2   ADH de l'octet d'adresse (opérande)
```

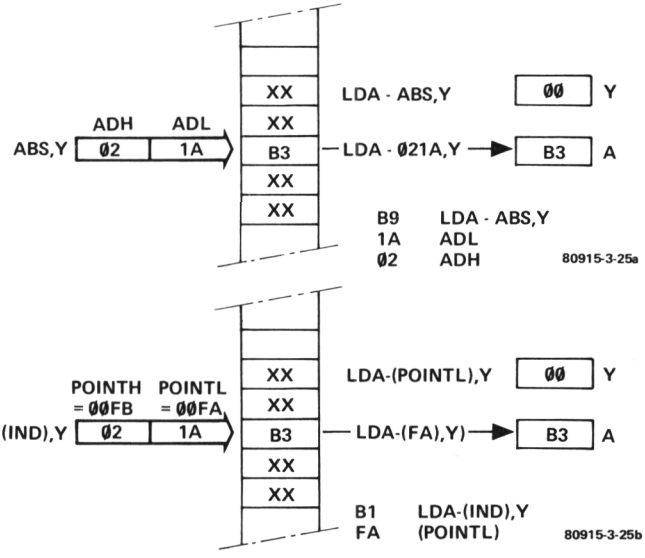
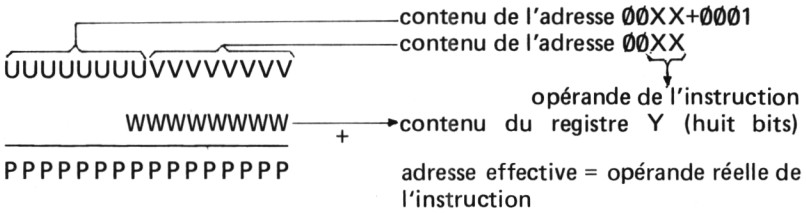


Figure 25. Chargement d'une donnée déterminée grâce à l'adressage indexé Y absolu (25a), et à l'aide de l'adressage indexé Y indirect (25b).

Après quoi, le contenu de l'adresse Ø21A (B3) est déposé dans l'accumulateur. Comme Y = ØØ, l'adresse de l'opérande est Ø21A + ØØ = Ø21A.
L'opération de la figure 25a exige que l'adresse de l'opérande soit connue, alors que pour celle de la figure 25b, qui concerne l'adressage indexé indirect, il en va autrement.
Avec une instruction dont l'adressage est indirect et qui comporte une partie d'une adresse servant d'opérande, la donnée sur laquelle l'instruction doit opérer *n'est pas* présente à l'adresse opérande. Elle est présente en un emplacement mémoire dont l'adresse se trouve à l'emplacement mémoire indiqué par l'adresse opérande.
Il nous faut donc être très attentifs et faire la différence entre *l'adresse opérande* et *l'adresse effective* ; les deux emplacements mémoire correspondant à l'adresse de l'opérande contiennent l'adresse (effective) de l'emplacement mémoire où est stockée la donnée sur laquelle l'instruction va opérer.
Revenons à la figure 25b. L'instruction qui va provoquer le chargement du contenu de l'emplacement mémoire Ø21A dans l'accumulateur est une instruction dont l'adressage est indexé indirect. La notation générale est: mnémonique-(IND), Y. Dans le cas de la figure 25b, c'est LDA-(FA),Y; l'information relative à l'adresse de l'opérande, et qui, dans ce contexte, est appelée également *l'adresse indirecte*, vient s'inscrire entre les parenthèses à la place de IND. FA est l'octet de droite de l'adresse indirecte. L'octet gauche de celle-ci est ØØ, ce qui indique que l'emplacement mémoire est situé sur la page zéro. Désignons l'adresse ØØFA par POINTL. A cette adresse est emmagasiné l'octet de droite de l'adresse effective, donc l'octet de gauche est stocké à l'emplacement mémoire faisant immédiatement suite, c'est-à-dire, à l'adresse ØØFB de la page zéro. Nous la désignons par POINTH.
La figure 25a nous montre que l'adresse effective doit être Ø21A; par conséquent, le contenu de POINTL est 1A et celui de POINTH est Ø2.
Nous constatons que FA fait suite au code opération de l'instruction (dans le cas présent, LDA-). Il en résulte qu'un seul des quatre octets désignant deux adresses où réside l'adresse effective, doit être précisé. Deux sur quatre sont automatiquement connus par le biais de l'adressage en page zéro, le troisième doit être spécifié après le code opération (FA). Quant au quatrième, c'est tout simplement l'emplacement mémoire suivant le troisième. Le microprocesseur sait qu'il se trouve là.
Mais ce n'est pas tout. N'oublions pas qu'il ne s'agit pas seulement d'adressage indirect, mais aussi, d'adressage indexé. Voyons donc comment il opère dans le cas qui nous intéresse.
Les deux emplacements successifs de la page zéro contiennent l'adresse effective, ainsi que nous l'avons dit. C'est précisément où se trouve le contenu (ØØ) du registre Y, ce que nous montre la figure 25b. En fait, l'adresse effective résulte finalement de l'addition du contenu des deux emplacements de la page zéro avec le contenu du registre Y.

Il est possible de donner une illustration graphique du processus:



Dans le cours de cette addition, une éventuelle retenue (carry) résultant de l'addition des octets V et W sera ajoutée à l'octet U.
Bien que deux adresses soient nécessaires (ØØXX et ØØXX + ØØØ1) à la détermination de l'adresse effective, il est toujours fait mention de *l'adresse* indirecte et non pas *des adresses* indirectes. A l'adresse indirecte réside une adresse absolue ou directe, qui, après addition avec Y, se transforme en adresse effective.
Les instructions de l'adressage indexé indirect comportent deux octets. Le premier d'entre eux contient le code opération, le second l'octet droit d'une adresse de la page zéro. Une liste de ces instructions avec leur code opération est donnée à la fin du livre.

Transfert de bloc de données avec adressage indexé indirect

Vous vous souvenez certainement que l'adressage absolu indexé X nous a permis de réaliser le transfert d'un bloc de données d'un champ à un autre. L'utilisation de ce programme autorisait le transfert d'un maximum de 256 octets. Mais il est fréquent qu'il soit nécessaire de transférer plus de 256 octets d'une section mémoire à une autre. Un petit sous-programme, BLMOVE, va nous permettre de transférer un maximum de 255 blocs de données de 256 octets chacun, soit un total de 255 x 256 = 65280 octets. C'est là un bon exemple de ce qu'il est possible de réaliser grâce à l'adressage indexé indirect Y.
La figure 26 montre le transfert de deux blocs de chacun 256 octets d'une section mémoire à une autre.
La première adresse du premier bloc est Ø2ØØ, la dernière Ø2FF. L'adresse de début (Ø2ØØ), adresse directe ou absolue, devrait être transférée dans deux adresses successives de la page zéro, et plus précisément aux adresses BEG ØØ et BEG+1 Ø2. Le premier bloc (qui est emmagasiné à la page Ø2) doit être transféré à la page A8 sur laquelle l'adresse de début est A8ØØ et l'adresse finale A8FF. La première adresse (A8ØØ), adresse directe ou absolue, devrait être transférée aux deux adresses successives de la page zéro, MOV ØØ et MOV +1 A8. Voici les diverses adresses de la page zéro ainsi que la désignation des octets y étant emmagasinés:

BEG	à l'adresse ØØØØ. Le contenu s'appelle FRADL, il est égal à ØØ
BEG+1	à l'adresse ØØØ1. Le contenu s'appelle FRADH, il est égal à Ø2
MOV	à l'adresse ØØØ2. Le contenu s'appelle TOADL, il est égal à ØØ
MOV+1	à l'adresse ØØØ3. Le contenu s'appelle TOADH, il est égal à A8
BLOCKS	à l'adresse ØØØ4. Le contenu s'appelle N, il est égal à Ø2

La valeur Ø2 inscrite à l'emplacement mémoire BLOCKS indique le

nombre de blocs de données à transférer. FR signifie: venant de, TO signifie: allant à, ADL signifie: octet droit, ADH signifie: octet gauche.
Le chargement des emplacements mémoire peut intervenir au moyen du clavier, mais un programme spécial serait tout de même bienvenu. C'est la raison d'être de DEFMOV, un programme (figure 27) qui charge les deux adresses indirectes BEG & BEG+1 et MOV & MOV+1 avec les valeurs correctes des données. Entrons maintenant dans le détail des opérations.
Le registre X est programmé en compteur de blocs et il est chargé avec le contenu de l'emplacement mémoire BLOCKS. Le registre Y fonctionne en authentique registre index. Son contenu désigne le rang de l'emplacement mémoire adressé dans un bloc de données de 256 octets. A supposer que le contenu de Y soit 21, le 33ème emplacement mémoire du bloc sera adressé. Au commencement du sous-programme BLMOVE, le contenu de Y est ØØ. Le transfert des données s'effectue par l'entremise de l'accumulateur. L'instruction pour le copiage des données est LDA, celle pour l'écriture est STA, s'agissant de l'adressage indexé indirect.
Après le transfert d'une donnée, le contenu de Y est décrémenté de 1 (DEY), ce qui permet au microprocesseur d'avoir accès à un nouvel emplacement mémoire. Le contenu de Y est successivement ØØ, FF, FE, . . . Ø2, Ø1, ØØ, FF, FE et ainsi de suite. Grâce à une instruction de branchement BNE, le programme vérifie que tous les octets d'un bloc ont été transférés. Si tel est le cas, le contenu de BEG+1 et de MOV+1 est incrémenté de 1 tandis que le contenu de X est décrémenté d'une unité par l'instruction DEX. La boucle LOOP du programme est donc parcourue 256 x le contenu de BLOCKS. Après cela, une instruction RTS permet le retour vers le sous-programme DEFMOV suivi d'une instruction BRK. Le programme est achevé.
Précisons encore quelques points. Les adresses indirectes BEG et BEG+1, ainsi que MOV et MOV+1, contiennent les adresses directes, qui, avec la prise en compte du contenu du registre index Y, sont nécessaires aux instructions de transfert à l'intérieur de la boucle LOOP. Il s'agit toujours de deux adresses consécutives de la page zéro. A l'avenir, les adresses telles que BEG, BEG+1 etc, seront désignées comme étant des *pointeurs d'adresses.* Leur format est de 16 bits et résulte de l'assemblage de deux octets, le contenu de l'adresse indirecte. Le pointeur est dirigé vers l'adresse directe ou absolue formée du contenu de l'adresse indirecte. Il existe différents types de pointeurs. La figure 26 montre les pointeurs BEG+1, BEG, MOV+1 et MOV, et l'on constate que la *notation d'un pointeur* réunit deux adresses séparées par une virgule. Ils pointent vers l'adresse de début d'un champ. Dès que le transfert d'un bloc est achevé, leur contenu est incrémenté de 1 et ils se déplacent de 256 positions. Tant que le transfert n'est pas terminé, leur position ne bouge pas et l'on peut dire qu'ils sont relativement statiques.
Il existe également ce que l'on pourrait appeler des pointeurs d'adresses dynamiques. Ils pointent vers les adresses dont le contenu dépend de l'état du registre Y et leur position change durant le transfert d'octet. La figure 26 montre les pointeurs dynamiques représentés par la dernière adresse des divers champs (Ø2FF, Ø3FF, A8FF, A9FF). Mais ceci pourrait prêter à confusion, car, il n'y a pas quatre pointeurs dynamiques, mais

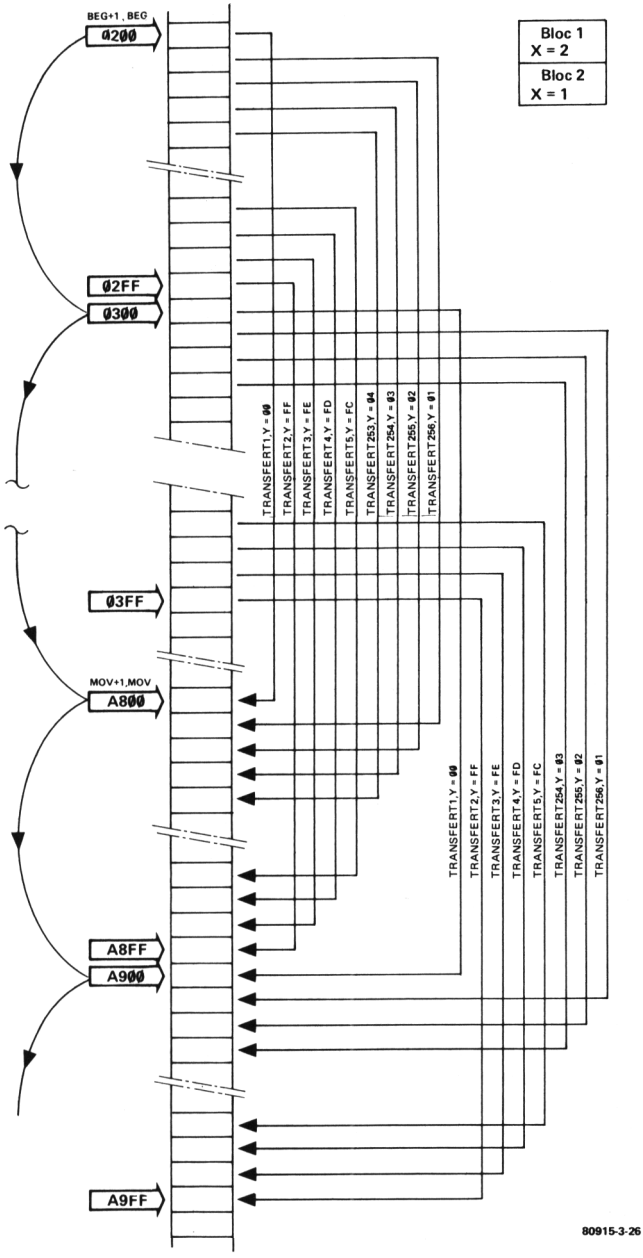


Figure 26. Transfert de deux blocs de chacun 256 octets. Le programme est donné en figure 27.

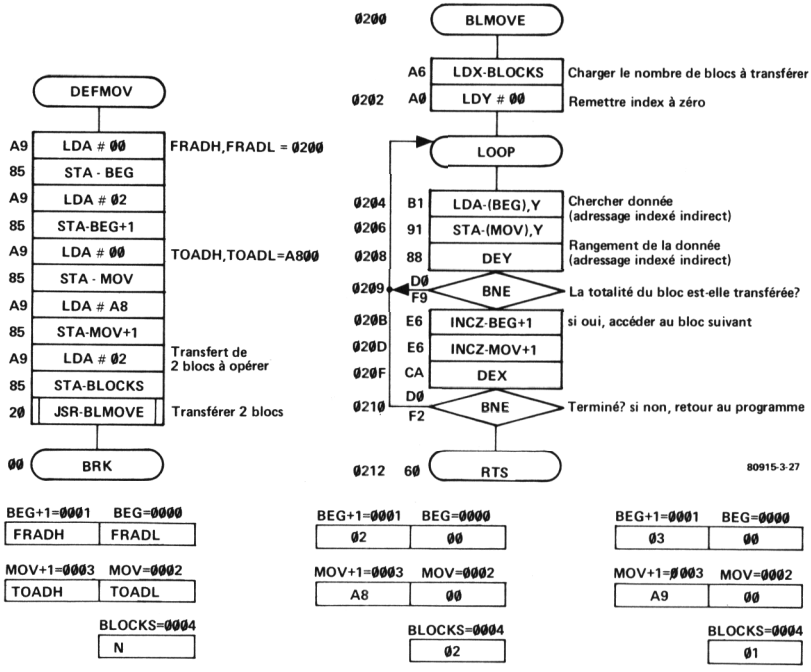


Figure 27. Programme à l'aide duquel un nombre maximal de 255 blocs de chacun 256 octets peut être transféré par l'utilisation de l'adressage indexé Y indirect.

deux; l'un pour le champ de départ et l'autre pour le champ d'arrivée. Ce qui signifie, qu'à un moment donné, les pointeurs dynamiques sont dirigés vers 02FF et A8FF, et l'instant d'après vers 03FF et A9FF.
Le pointeur dynamique du champ de départ pointe vers l'adresse de l'opérande de l'instruction LDA; celui du champ d'arrivée pointe vers l'adresse de l'opérande de l'instruction STA.

Adressage indirect indexé X

Vous avez certainement noté l'inversion entre indexé et indirect dans l'énonciation des modes d'adressage; à cela s'ajoute le remplacement de Y par X. Ces interversions signifient qu'il s'agit d'un mode d'adressage comportant la détermination de l'adresse indirecte, laquelle contient l'adresse effective indiquant l'opérande réelle de l'instruction.
Les deux modes d'adressage ont en commun le fait que les emplacements mémoire de la page zéro servent de pointeurs d'adresses et que l'instruction comporte deux octets. Différence essentielle, l'adresse indirecte réelle est formée du second octet de l'instruction et du contenu du registre X; autrement dit, au second octet est ajouté l'octet représentant le contenu du registre index X. Une éventuelle retenue sera ignorée, ce qui indiquera que les calculs sont terminés et que l'octet de droite de la première moitié

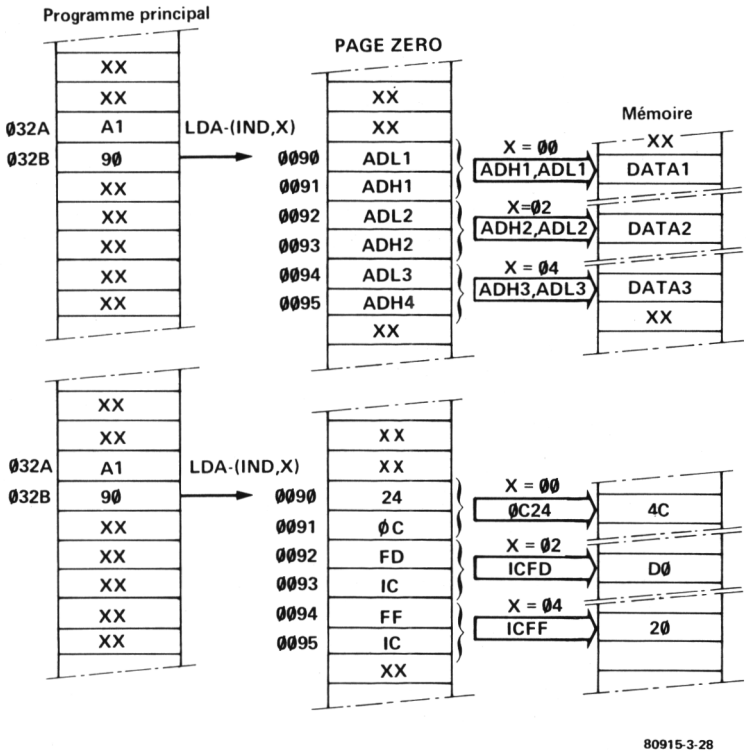


Figure 28. Exemple d'adressage indirect indexé X. C'est un mode d'adressage peu utilisé.

de l'adresse indirecte est déterminé, l'octet gauche étant ØØ, en raison de l'adressage page zéro. La seconde moitié de l'adresse indirecte (dont l'octet gauche contient l'adresse effective) suit automatiquement puisque c'est l'adresse de l'emplacement mémoire suivant de la page zéro.

Concordances et différences entre les deux modes d'adressage peuvent se résumer de la manière suivante:

- dans l'adressage indexé Y indirect, l'adresse indirecte est déterminée directement par le second octet de l'instruction; l'adresse effective résulte indirectement du contenu de l'adresse indirecte, laquelle comporte le contenu du registre Y.
- Dans l'adressage indirect indexé X, l'adresse indirecte résulte indirectement de l'addition, au second octet de l'instruction, du contenu du registre X; l'adresse effective découle directement du contenu de l'adresse indirecte.

Le programme présenté en figure 28 devrait nous faire comprendre le mécanisme du déroulement d'une instruction LDA à l'aide de l'adressage indirect indexé X. Nous distinguons trois sections de mémoire illustrant

le processus général. La section de gauche (MAIN PROGRAM = Programme principal) présente quelques emplacements mémoire faisant partie du programme, celle du milieu une partie de la page zéro, et celle de droite (MEMORY) les emplacements mémoire où pourraient être emmagasinées les données à charger.
Le code opération de l'instruction LDA est stocké à l'adresse Ø32A du MAIN PROGRAM. La notation s'écrit: LDA- (IND,X). S'il s'état s'agi de l'adressage indexé Y indirect, c'eût été LDA-(IND), Y, avec le code opération B1. A l'adresse suivante, Ø32B est stocké 9Ø. A celle-ci va être ajouté le contenu de X. Supposons que celui-ci soit ØØ, l'adresse effective se trouvera aux emplacements mémoire ØØ9Ø (ADL1) et ØØ91 (ADH1) de la page zéro. Le pointeur d'adresse ADH1, ADL1 est pointé vers l'adresse de la section MEMORY où est inscrite DATA1 (Donnée 1). DATA1 est déposée dans l'accumulateur.
Admettons que le contenu de X soit Ø1. Nous nous retrouvons aux emplacements mémoire ØØ91 et ØØ92 où sont stockés respectivement ADH1 et ADL2, octets d'adresse ne correspondant pas aux mêmes emplacements mémoire. Ce qui n'est jamais prévu. Cela ne peut arriver que si le contenu du registre X est Ø2 ou en tout cas, la modification ne peut intervenir qu'avec des résultats pairs.
Si le contenu de X est Ø2, l'adresse effective est stockée aux emplacements mémoire ØØ92 et ØØ93; le pointeur d'adresse ADH2, ADL2 est pointé vers DATA2 (Donnée 2). Lorsque X = Ø4, l'adresse effective est emmagasinée aux adresses ØØ94 et ØØ95 et le pointeur d'adresse ADH3, ADL3 est pointé vers DATA3 (Donnée 3) de la section MEMORY.
L'intérêt de cette méthode d'adressage réside dans le fait qu'ainsi la page zéro de la mémoire de travail peut être utilisée en tant que "Pointer Look Up Table" (Tableau indicateur des pointeurs), ce qui veut dire que, en page zéro, peuvent être déterminés des pointeurs d'adresses pointés vers des emplacements mémoire précis.
Cependant, c'est un mode d'adressage relativement peu employé, mais sur lequel nous reviendrons dans un autre chapitre de ce livre où nous traiterons plus particulièrement du bloc I/O. La liste des instructions de l'adressage indirect indexé X est présentée à la fin de cet ouvrage.

Interruption d'un programme en cours

Instructions NMI, IRQ et RESET, et adressage indirect

Dans le cours du paragraphe "Les interruptions" du chapitre 1, nous avions déjà abordé le sujet; le moment est venu de l'étudier plus en détail.
Une interruption d'un programme en cours résulte de l'émission d'un signal (niveau de tension ou impulsion) par un organe d'entrée/sortie. L'exécution du programme s'arrête à l'endroit où le microprocesseur en était arrivé au moment de la réception du signal. A partir de ce point, l'ordinateur saute à un sous-programme qui exécute l'opération ayant fait l'objet de la demande d'interruption. Dès l'achèvement du sous-programme, l'ordinateur reprend le cours du programme interrompu.
Lorsque la demande d'interruption est reçue par l'ordinateur, et avant que celui-ci ne quitte le programme en cours pour exécuter le sous-programme

correspondant à la requête, il est nécessaire qu'un certain nombre d'informations relatives au programme principal soient sauvegardées, ainsi que le contenu du compteur ordinal, PC, et celui du registre d'état, P, les unes et les autres étant déposés dans la pile (Stack).
C'est donc un signal déterminé appliqué à une broche du microprocesseur qui va provoquer le saut à un sous-programme, en réponse à une demande d'interruption. Par conséquent, il faut bien réaliser que l'opération ne résulte pas de la mise en œuvre d'une certaine combinaison d'octets (le code opération de JSR), mais de l'intervention d'une tension électrique; en somme, du matériel et non pas du logiciel.
Il existe deux possibilités d'appliquer un signal en cours de déroulement de programme:

1. *Utilisation de la requête d'interruption, IRQ*
   Si la tension présente à la broche IRQ du microprocesseur est à un état logique signalant que le programme peut être interrompu, la requête d'interruption *pourra* être satisfaite. Mais, il est également possible d'ignorer la demande, par l'intermédiaire du logiciel, et ceci résulte de l'état de l'indicateur d'interruption I (Interrupt flag) conditionné par l'une des deux instructions:
   CLI (code opération 58), grâce à quoi I = Ø, ce qui signifie que la demande d'interruption est validée (Interrupt Enable)
   ou
   SEI (code opération 78), grâce à quoi I = 1, ce qui signifie que la demande d'interruption est inhibée (Interrupt Disable) et qu'aucune interruption n'est possible via IRQ.
2. *Interruption non masquable, NMI*
   Si l'état logique de la tension présente à la broche NMI du microprocesseur le permet, le programme en cours *doit* être interrompu et le microprocesseur saute à un sous-programme d'interruption afin de satisfaire la demande.

Les signaux d'interruption IRQ et NMI, ainsi que le signal RES qui sera étudié ultérieurement, peuvent être considérés comme un genre d'instructions de saut hardware (provenant d'équipements matériels externes), ayant un caractère inconditionnel dans le cas de NMI, ou conditionnel dans le cas de IRQ. Du point de vue du hardware (matériel), il existe une autre différence entre ces deux possibilités d'interruption. Une instruction NMI est exécutée par le passage de l'état logique "1" à l'état logique "Ø" de l'entrée NMI; c'est-à-dire pendant le passage rapide de la tension électrique vers 0 volt. Pour assurer l'exécution d'une instruction IRQ, il faut que la broche IRQ soit au niveau logique "Ø". Une interruption (IRQ) ne peut donc avoir lieu qu'entre le passage 1 à Ø et le passage de Ø à 1 suivant.

Exécution d'une instruction NMI

Dès qu'une instruction NMI intervient, le microprocesseur interroge les emplacements mémoire aux adresses FFFA et FFFB afin de détecter en quel emplacement de la mémoire débute la routine d'interruption. La figure 29 nous montre que l'adresse est 2ØØØ. L'octet droit de celle-ci est stocké en FFFA et le gauche l'est en FFFB. A l'aide de ces deux octets, le microprocesseur assemble le *vecteur NMI* qui n'est autre que le pointeur

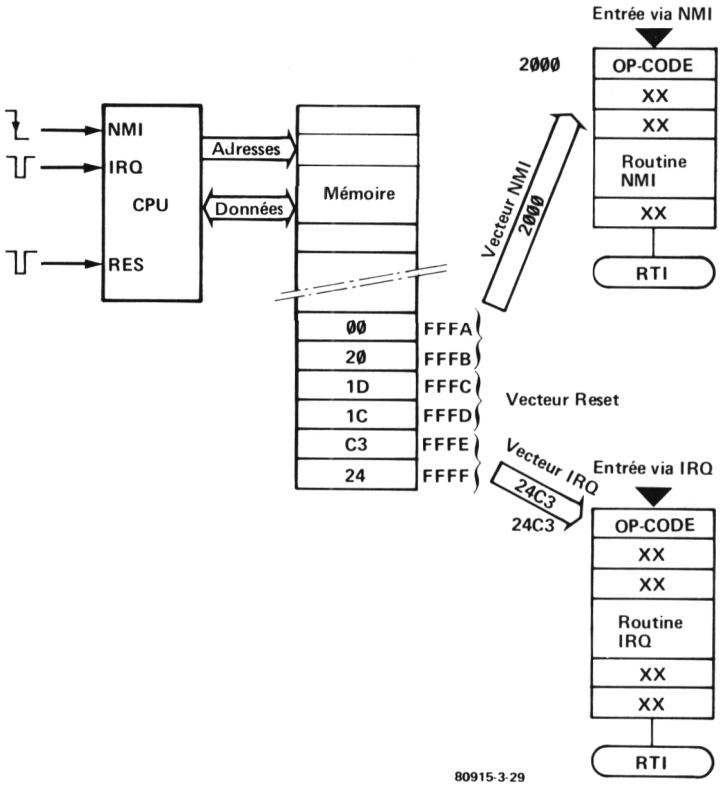


Figure 29. Après une instruction NMI ou (parfois, car conditionnelle) après une instruction IRQ, un saut est effectué à un sous-programme qui est achevé. Le vecteur NMI pointe vers l'adresse de début du sous-programme NMI, le vecteur IRQ pointe vers l'adresse de début du sous-programme IRQ. Le retour au programme principal s'opère grâce à une instruction RTI (et *non pas* RTS).

d'adresse, dont nous avons déjà fait usage, et qui pointe vers l'adresse de début du sous-programme. De la même façon que dans l'adressage indexé indirect, le contenu du pointeur d'adresse est emmagasiné au bas de la page zéro, le contenu du vecteur NMI est déposé au bas de la page FF.
Ensuite, le sous-programme est exécuté de bout en bout; il faut noter qu'il comporte plusieurs branchements vers d'autres sous-programmes. Une instruction RTS met fin à un sous-programme normal; dans le cas d'une routine d'interruption, c'est une instruction RTI, dont le code opération est 40, qui ramène le Junior Computer à l'emplacement mémoire où il reprend le programme principal.

Exécution d'une instruction IRQ

Après que la broche IRQ soit venue à l'état logique 0 et pendant tout le temps où celle-ci y reste, une demande d'interruption est appliquée au processeur.

Dès lors, l'ordinateur vérifie l'état de l'indicateur d'interruption I dans son registre d'état. Si l'indicateur est à "1", la demande est ignorée et le programme en cours continue à se dérouler. Par contre, s'il est à "Ø", l'interruption est autorisée, le microprocesseur saute à la routine IRQ. A l'adresse de début de celle-ci, *le vecteur IRQ* pointe vers les emplacements mémoire FFFE et FFFF (les octets droit et gauche, respectivement). Dans l'exemple de la figure 29, il s'agit de l'adresse 24C3. L'instruction RTI met fin à la routine d'interruption et le microprocesseur reprend l'exécution du programme principal. Il est évident que, dans le cadre de la routine d'interruption, l'ordinateur peut être branché sur un certain nombre de sous-programmes.
Dès qu'est engagée la routine d'interruption (l'indicateur I étant à "Ø"), I passe à l'état "1" afin de garantir que le sous-programme ne soit pas repris dès son achèvement.
La mise à "Ø" de l'indicateur I intervient par l'intermédiaire d'une instruction CLI ou RTI. Il faut veiller à ce que la ligne IRQ du processeur soit remise à "1" (reset) avant que l'indicateur I ne soit lui-même aussi remis à "1". Si tel n'était pas le cas, l'instruction IRQ serait exécutée une seconde fois.
L'instruction IRQ pouvant être ignorée, alors que l'instruction NMI ne peut l'être, il en résulte que cette dernière a une *priorité supérieure* à celle de la première.
La question se pose de savoir comment, après l'exécution d'une routine d'interruption, le microprocesseur reprend le déroulement du programme principal au point même où il l'avait abandonné. La réponse est très simple. Souvenons-nous de l'instruction JSR. Lorsque, dans le déroulement du programme principal, le microprocesseur rencontre cette instruction, il sauvegarde le contenu du registre d'état P (le "registre des indicateurs"), ainsi que celui du compteur ordinal PC, et il dépose le contenu de l'adresse, à laquelle il a interrompu le programme principal, sur la pile. C'est seulement après ces mesures de sauvegarde qu'est opéré le branchement sur un sous-programme. A la fin de celui-ci, une instruction RTS ramène le microprocesseur à l'adresse précédemment déposée sur la pile, à partir de laquelle il reprend le déroulement du programme principal. Une séquence d'interruption se déroule suivant le même schéma. L'instruction d'interruption joue le même rôle que l'instruction JSR, et l'instruction RTI a une action semblable à celle de l'instruction RTS. Le registre d'état et le compteur ordinal sont restaurés et le pointeur de pile est remis à jour.

Instruction de Reset

La broche RES du microprocesseur offre une autre possibilité d'action sur son fonctionnement, dans lequel intervient le logiciel par l'intermédiaire du matériel. Lorsque cette broche reste pendant un temps suffisamment long à l'état "Ø", le programme saute à un emplacement mémoire vers lequel pointe le *vecteur reset*; les adresses des octets composant ce vecteur sont FFFC (octet droit) et FFFD (octet gauche).
Dans le Junior Computer, le vecteur reset pointe toujours vers l'adresse 1C1D. C'est à partir de celle-ci que débute le programme moniteur. Après mise sous tension de l'ordinateur, si l'on presse, par exemple, la touche RST, une bascule est mise à l'état "Ø" et porte la broche RES à l'état

"Ø". Ainsi que nous venons de le voir, le vecteur reset pointe vers 1C1D qui est l'adresse de début du programme moniteur du système, lequel gère l'affichage et interroge le jeu de touches du clavier.
Le contenu des trois vecteurs est stocké à la page FF. Celle-ci contient les adresses les plus élevées. Se pose alors la très intéressante question de savoir comment le microprocesseur peut adresser les vecteurs d'interruption et de reset aux emplacements mémoire FFFA . . . FFFF, si l'on se souvient que le Junior Computer a une capacité de décodage limitée à 8 x 4 = 32 pages allant de ØØ à 2Ø. (Voir au chapitre 1 le paragraphe "Organisation de la mémoire").
Heureusement, le problème des emplacements mémoire non branchés peut être résolu. L'octet de gauche d'un emplacement mémoire de vecteur est FF. En langage binaire, cela se traduit par 11111111, ces bits étant déterminés par les lignes d'adresses (de la gauche vers la droite) A15 . . . A8. Mais, précisément en raison de la capacité de décodage limitée de la version standard du Junior Computer, les lignes d'adresses A15, A14 et A13 ne sont reliées, à l'intérieur du système, à aucun élément et il n'y a donc pas d'importance à ce que leur état soit à "Ø" ou à "1". Dans ce cas, 11111111 est égal à XXX11111, et, par exemple, à ØØØ11111, ce qui équivaut à 1F. En d'autres termes, le Junior Computer utilise la page 1F comme s'il s'agissait de la page FF. Quant à la page 1F, elle fait partie avec les pages 2Ø, 1E et 1D, de l'EPROM d'une capacité de 1K. L'EPROM convient parfaitement au stockage des vecteurs.

La figure 30, qui représente quelques sections de mémoire, nous permettra d'observer de plus près le déroulement d'une instruction IRQ ou NMI. Nous constatons que la page Ø1 sert de pile; la page Ø3 abrite le programme qui va être interrompu. En outre, le vecteur IRQ pointe vers 24C3 qui est l'adresse de début de la routine IRQ, et le vecteur NMI pointe vers l'adresse 2ØØØ où débute la routine NMI.
A un moment donné du déroulement du programme principal, le microprocesseur atteint l'adresse Ø343 où est stocké le code opération d'une instruction. A cet instant, une instruction NMI est émise; c'est une demande d'interruption qui va mobiliser immédiatement le micro-ordinateur. Sans aller au fond des détails sachons que l'octet gauche (PCH) du compteur ordinal (Ø3) est déposé sur la pile à l'emplacement vers lequel pointe le pointeur de pile. En fait, il s'agit de l'emplacement vide immédiatement au sommet de la pile. Ensuite, le pointeur est incrémenté de 1 et l'octet droit (PCL) est chargé au sommet de la pile. Après avoir été incrémenté de 1, le pointeur est dirigé vers l'adresse Ø1FC. Le contenu du registre d'état est ensuite déposé sur la pile et le pointeur de pile pointe alors vers Ø1FB.
Intervient ensuite le saut au sous-programme NMI. Celui-ci débute à l'adresse dont le contenu indique la direction du vecteur NMI; dans le cas présent, il s'agit de 2ØØØ. L'indicateur d'interruption I est incrémenté de 1, grâce à quoi une éventuelle instruction IRQ ne sera pas validée. Après que le contenu du vecteur NMI ait été chargé dans le compteur ordinal, la routine d'interruption NMI est exécutée.
Revenons maintenant à l'adresse Ø343 et imaginons que ce soit, non pas une instruction NMI, mais une demande d'interruption IRQ. Les opéra-

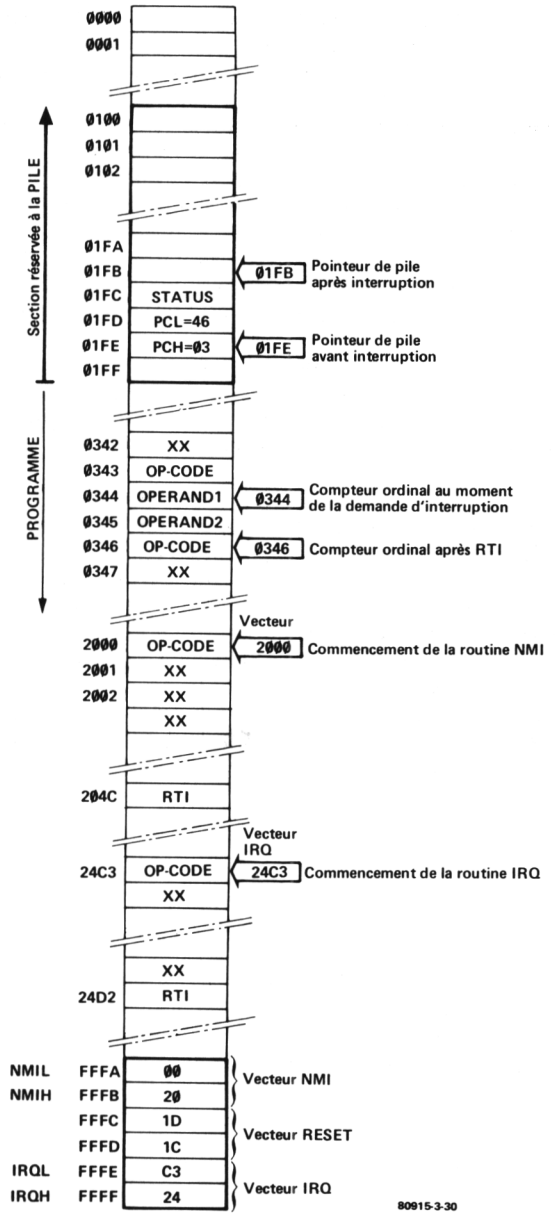


Figure 30. Une carte de plusieurs sections de mémoire permet d'observer le déroulement des routines d'interruption.

tions vont se dérouler de manière assez semblable puisque PCH, PCL et le contenu de P sont empilés dans l'ordre. Nous savons que la routine IRQ ne peut être exécutée que si l'indicateur d'interruption I est à "0". Le contenu du vecteur IRQ sera prélevé aux adresses FFFE et FFFF, ce qui donnera l'adresse à laquelle débute le sous-programme. Ensuite, l'indicateur I est remis à "1" afin que toute nouvelle demande d'interruption IRQ soit ignorée. Mais que se passerait-il s'il s'agissait d'une demande NMI, qui, ne l'oublions pas, a une priorité absolue? Et qu'en serait-il si plusieurs NMI étaient émises? Nous aborderons ce problème ultérieurement à l'occasion de l'étude de l'instruction JMP avec l'adressage indirect. Pour le moment, contentons-nous de noter que le microprocesseur peut exécuter *plusieurs* instructions NMI grâce à *un* vecteur NMI.
De la même manière que pendant l'exécution d'un sous-programme, on peut faire appel à un autre sous-programme, on peut aussi faire appel, pendant l'exécution d'un sous-programme d'interruption, à un autre sous-programme d'interruption (interrupt nesting).

L'exécution d'une instruction RTI

Cette instruction ramène le microprocesseur au programme principal après l'achèvement d'une routine d'interruption. Dans le programme illustré par la figure 30, il y a une instruction RTI à l'adresse 204C. D'abord, le pointeur de pile est décrémenté de 1 et pointe vers l'adresse 01FC où est stocké le contenu du registre d'état, P, et le contenu de ce registre modifié (en fonction du déroulement de la routine d'interruption) est copié dans le microprocesseur.
Après une nouvelle décrémentation du contenu du pointeur de pile de 1 (01FD), le compteur ordinal PCL reprend sa valeur primitive; ensuite, après une décrémentation supplémentaire du pointeur (01FE), PCH retrouve lui-aussi sa valeur de départ. L'adresse de retour est reconstituée, 0346, dans le cas de la figure 30. Le microprocesseur est revenu au programme principal.

La sauvegarde des registres

Lorsque le microprocesseur saute à une routine d'interruption, le contenu du registre P ("registre des indicateurs") et celui du compteur ordinal (adresse de retour) doivent être sauvegardés. Fréquemment, il est nécessaire de préserver d'autres informations comme, par exemple, le contenu des registres internes A, X et Y. C'est possible grâce à des instructions appropriées qui déposent le contenu de ces registres sur la pile. Bien entendu, il doit être possible de les y reprendre dès que c'est nécessaire, grâce à d'autres instructions.
La routine d'interruption concernée reçoit au début de son déroulement un certain nombre d'instructions de sauvegarde, contrebalancées, en fin de programme, par d'autres instructions de prélèvement. Par exemple:

```
SAVE  PHA  sauvegarder le contenu de l'accumulateur sur la pile (après
           quoi, le pointeur de pile est incrémenté de 1)
      TXA  transférer le contenu de X dans l'accumulateur
      PHA  sauvegarder le contenu de l'accumulateur sur la pile (après
           quoi, le pointeur de pile est incrémenté de 1)
```

```
        TYA   transférer le contenu de Y dans l'accumulateur
        PHA   sauvegarder le contenu de l'accumulateur sur la pile (après
              quoi, le pointeur de pile est incrémenté de 1)
        . . . première instruction de la routine d'interruption
          .
          .
          .
RESTO   PLA   transférer le sommet de la pile dans l'accumulateur (après
              quoi, le pointeur de pile est décrémenté de 1)
        TAY   transférer le contenu de l'accumulateur dans Y
        PLA   transférer le sommet de la pile dans l'accumulateur (après
              quoi, le pointeur de pile est décrémenté de 1)
        TAX   transférer le contenu de l'accumulateur dans X
        PLA   transférer le sommet de la pile dans l'accumulateur (après
              quoi, le pointeur de pile est décrémenté de 1)
              (restaure le contenu primitif de l'accumulateur)
END     RTI   retour au programme initial interrompu
```

Il est bon de noter que le vecteur IRQ ou NMI reste pointé vers l'adresse de SAVE, et non vers l'adresse de début de la routine d'interruption concernée. Souvenons-nous également que la première donnée posée sur la pile est la dernière à y être prélevée (structure LIFO).

Le treizième et dernier mode d'adressage est l'adressage indirect. Il résulte d'une seule instruction, JMP-indirect. Mais, cela nous amène à examiner comment il est possible d'exécuter plusieurs routines NMI, alors qu'un seul vecteur NMI est stocké aux adresses FFFA et FFFB.

Adressage indirect

Le mode d'adressage absolu nous avait permis de nous familiariser avec l'instruction JMP, qui est l'instruction de saut inconditionnel à une autre partie de la mémoire où sont stockées d'autres instructions. Elle comporte trois octets. Le premier est celui du code opération (4C), le second est l'octet droit ADL et le troisième est l'octet gauche ADH de l'adresse vers laquelle le saut est exécuté.

L'instruction JMP du mode d'adressage indirect comporte, elle aussi, trois octets. Le premier est aussi celui du code opération (6C), le second et le troisième contiennent l'ADL et l'ADH d'une adresse d'un *vecteur-saut.* L'octet droit de ce vecteur est stocké à l'emplacement mémoire indiqué par le pointeur d'adresse ADH, ADL tandis que l'octet gauche est emmagasiné à l'emplacement suivant indiqué par le pointeur d'adresse (ADH,ADL) + ØØØØØØØ1. C'est ainsi qu'est déterminé le contenu du vecteur-saut et par conséquent l'adresse effective du saut vers laquelle il est pointé.

L'instruction est notée JMP-(IND), ce qui, dans le cas de la figure 31, se traduit pas JMP-(1A7A). L'octet de droite de l'adresse du saut est stocké à l'emplacement mémoire 1A7A et l'octet de gauche l'est à l'emplacement 1A7B. Il s'agit d'un saut indirect vers des adresses déterminant le vecteur-saut, et d'un *saut effectif* vers une adresse de saut vers laquelle le vecteur-saut pointe.

Quel rapport y a-t-il donc entre le saut indirect et l'exécution d'une instruction NMI? Revenons à l'examen de la figure 31, où nous voyons une instruction NMI. Le microprocesseur interroge le contenu des adresses FFFA et FFFB, à la recherche du vecteur NMI. Celui-ci semble être 1FAB, mais ce n'est pas l'adresse de début d'un sous-programme. C'est celle où est stocké le code opération de l'instruction JMP-(IND). Le contenu de IND est 1A7A (soit le contenu des adresses 1FAC et 1FAD). L'adresse qui permettra que soit effectué le saut (vers laquelle pointe le *vecteur-saut NMI* et qui n'est pas la même que celle du *vecteur-NMI*) est emmagasinée aux emplacements mémoire 1FAC et 1FAD de la page 1F.
La page 1F fait partie de la section RAM de la mémoire située dans le PIA (voir chapitre 1). Le vecteur-saut NMI est donc à l'emplacement opposé à celui du vecteur NMI (lequel est stocké sur la page FF, assimilée à la page 1F de l'EPROM) et il est spécifié par l'utilisateur. C'est absolument indispensable à l'exécution de *diverses* routines NMI grâce à *un seul* vecteur NMI et différents vecteurs-saut NMI. Mais, pour que le processus soit mené à bonne fin, il est nécessaire que, avant qu'une instruction NMI intervienne, les adresses indirectes soient chargées du contenu des vecteurs de saut, et par conséquent, que les adresses 1A7A et 1A7B soient chargées avec un vecteur-saut NMI, et les adresses 1A7E et 1A7F le soient avec un vecteur-saut IRQ. Les emplacements mémoire 1C1F et 1C2Ø accueillent le vecteur-saut RES. Les vecteurs-saut doivent pointer vers l'emplacement mémoire où est stocké le code opération de JMP-(IND), c'est-à-dire 6C.
Dans l'exemple de la figure 31, après l'intervention d'une instruction NMI et, par l'intermédiaire d'un vecteur NMI, d'une instruction JMP-(IND) et du vecteur-saut NMI, le programme revient à l'emplacement mémoire 1CØØ où est stocké le premier code opération de la routine d'interruption. Après qu'une demande IRQ ait été validée, le programme devra se retrouver à l'emplacement mémoire 2Ø24, par l'intermédiaire du vecteur IRQ, d'un JMP-(IND) et d'un vecteur-saut IRQ.
Ce qu'il y a de remarquable, c'est que l'adresse de saut effective d'un sous-programme d'interruption est variable; tantôt grâce à une intervention de l'utilisateur, tantôt sous l'influence du programme. Concernant ce dernier, une partie d'un sous-programme d'interruption déterminé devrait permettre que l'adresse indirecte soit chargée d'une nouvelle adresse de saut (directe, absolue) effective appartenant à une autre routine d'interruption.
Une instruction JMP de l'adressage indirect a encore une autre utilisation très intéressante, n'ayant rien à voir avec les interruptions. Il est possible également d'intégrer un tel saut dans un programme en se servant du code opération 4C au lieu d'une instruction de saut directe. L'avantage réside dans le fait que, lors du contrôle de programmes importants, le programmeur puisse écrire durablement son programme dans une EPROM, sans pour autant connaître les adresses directes de saut. Ainsi, ces adresses peuvent être inscrites dans des emplacements mémoire de RAM et sont donc modifiables selon les nécessités. Quand le programme écrit dans l'EPROM a été testé, les codes opérations des instructions JMP-(IND) peuvent être remplacés par des instructions normales JMP comportant, pour les sauts de programme directs et indirects, trois octets.

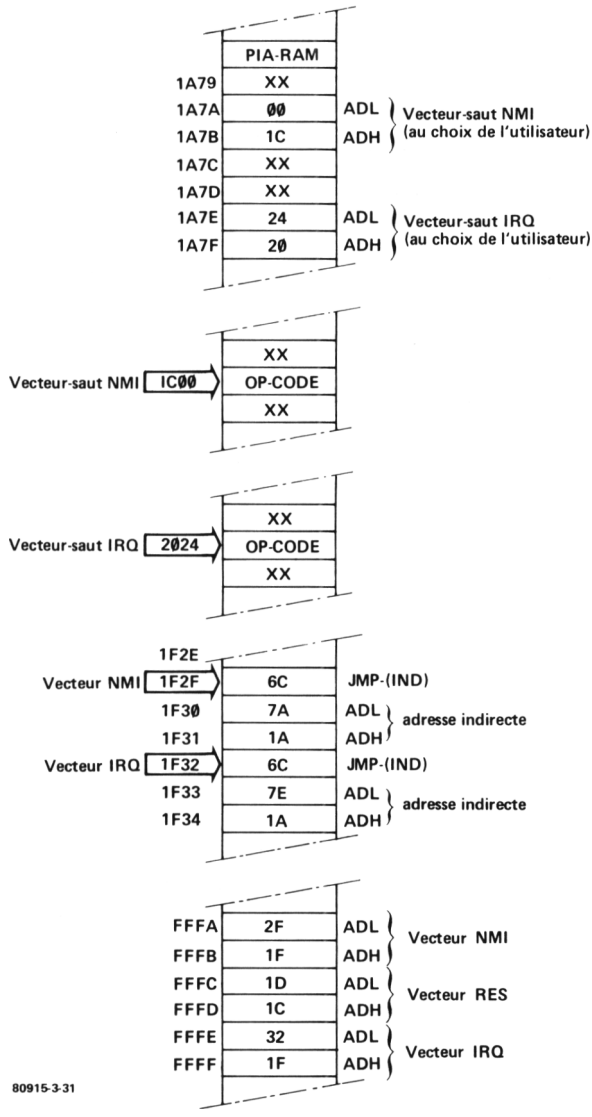


Figure 31. Traitement d'interruptions grâce à des instructions JMP avec mode d'adressage indirect. Le contenu de l'adresse indirecte indique l'adresse de saut (directe = absolue) effective et il est programmable.

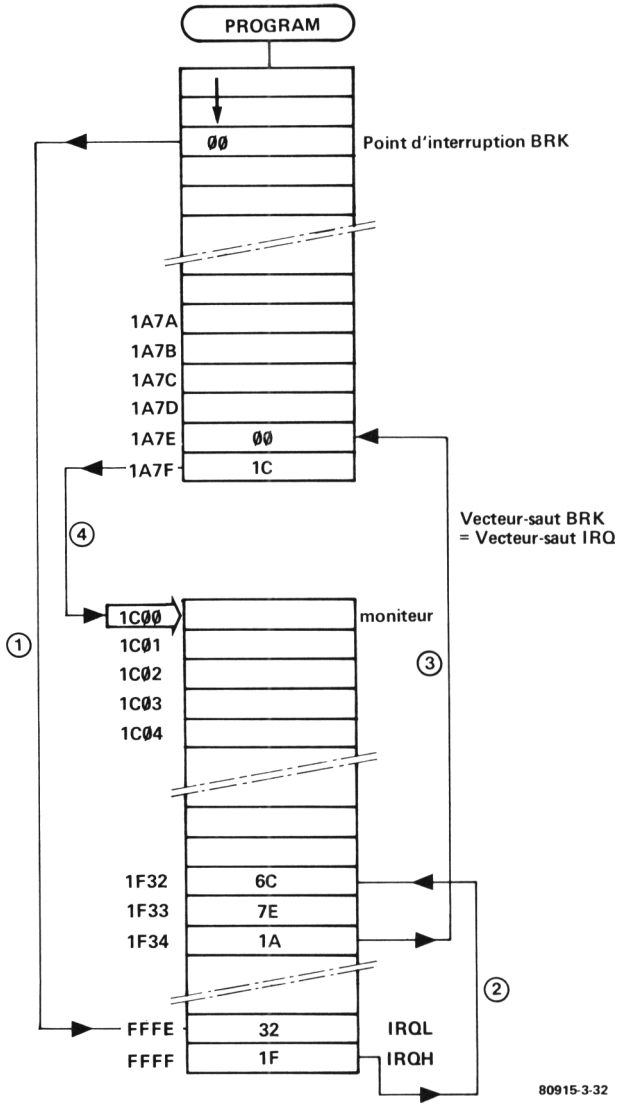


Figure 32. L'instruction BRK est la version software (logiciel) de l'instruction hardware (matériel) IRQ et elle est utilisable, entre autres, au déverminage ("debbuging") d'un programme.

L'instruction BRK

Jusqu'à présent, les interruptions de programme étudiées n'étaient possibles que grâce aux instructions "hardware" NMI et IRQ. Mais elles peuvent aussi bien être réalisées par le logiciel à la suite de l'instruction BRK, dont le code opération est ØØ, et que nous avons déjà rencontrée à la fin d'un programme.

Lorsque, dans le cours d'un programme, le code opération ØØ est lu dans la mémoire, la situation est équivalente à celle résultant d'une instruction IRQ. Après une instruction BRK, le microprocesseur détecte les adresses FFFE et FFFF afin de savoir où pointe le vecteur IRQ et donc à quelle adresse le programme doit sauter.

La figure 32 montre un exemple d'utilisation de l'instruction BRK. Le processeur reçoit une telle instruction à l'adresse 1A7D, et de là il part à la recherche du vecteur IRQ aux adresses FFFE et FFFF. Le programme continuant à se dérouler, le microprocesseur détecte ensuite l'adresse vers laquelle pointe le vecteur IRQ. La figure 32 nous montre, en outre, que la partie du programme débutant à l'adresse 1CØØ est baptisée TEST et que l'instruction BRK sépare une section correcte du programme d'une autre section n'opérant pas normalement. L'instruction BRK est extrêmement utile au "déverminage" (debugging) de cette dernière.

Une instruction BRK peut également provoquer le retour au programme moniteur, et précisément aux emplacements mémoire FFFE et FFFF où est stocké le vecteur IRQ.

L'instruction BRK comporte un indicateur (flag) faisant partie du registre P. Dès qu'intervient un BRK, l'indicateur B est mis à "1", car s'il s'agissait d'une instruction IRQ (hardware) l'indicateur B serait mis à "Ø".

Ainsi s'achève le chapitre 3 qui a comporté peu de pratique et beaucoup de théorie, même si nous avons fait souvent appel à des programmes-type. Les choses vont changer avec le chapitre 4 dans lequel nous traiterons du PIA, car, désormais, nous savons très bien ce que sont les interruptions et nous connaissons sur le bout des doigts les treize modes d'adressage du Junior Computer.

4

Pour débuter, des programmes simples

Programmation sans confusion

Nous voici arrivés au dernier chapitre de ce premier livre. Le second livre vous réserve des programmes extrêmement intéressants. Mais, pour en tirer le meilleur parti, il faut que vous commenciez par vous exercer à la programmation. C'est l'objet de ce chapitre dont la matière de base sera un certain nombre de programmes-types. Car, pour bien apprendre, rien ne vaut encore un bon exemple.

Le bloc d'entrées/sorties programmé, les routines du programme moniteur, l'éditeur et l'assembleur seront examinés en détail dans le cours du second livre consacré au Junior Computer. Ces sujets particulièrement attractifs ne seront cependant étudiés qu'après que vous ayez acquis la maîtrise de la programmation que vous conférera l'assimilation des connaissances fondamentales dispensées par ce quatrième chapitre et ceux qui l'ont précédé. Au fond, personne ne s'étonne que, par exemple, l'étude du piano débute, non par les sonates de Chopin, mais plus simplement par "Au clair de la lune". En tout cas, par des exercices sans complexité excessive.
Vous trouverez donc dans ce chapitre un certain nombre de programmes accompagnés chaque fois par un ordinogramme général et un autre sensiblement plus détaillé, cet ensemble étant complété par l'illustration de la frappe des touches, ainsi que ce fut déjà fait dans le cours du chapitre 3. En outre, nous vous ferons connaître quelques "astuces" pratiques.

Exercice de doigté sur le programme d'addition

C'est littéralement vrai, car, il vous faudra presser un total de 594 fois l'une ou l'autre des touches du clavier avant que le programme permettant l'addition décimale ait été inscrit dans la mémoire de travail (RAM) du Junior Computer, sous la forme de 196 octets occupant autant d'emplacements mémoire. Lorsque ce sera fait, il nous sera possible d'effectuer l'addition de deux nombres décimaux de six chiffres au plus.

Le programme d'addition a été exposé dans le cours du chapitre 3 où les figures 20, 21 et 14 donnent respectivement l'ordinogramme détaillé du programme principal, l'ordinogramme détaillé de neuf sous-programmes, les noms et adresses de chacun des trois tampons d'affichage. Les ordinogrammes détaillés constituent la dernière phase de la préparation du programme; chaque opération, chaque décision est traduite en instructions appropriées. Les codes opération, et, par conséquent, les octets d'adresses des emplacements mémoire sont connus, c'est une question de choix (voir annexe). Il en est de même des opérandes et des octets de déplacement, bien que ces données nécessaires à l'exécution des opérations ne soient pas toujours connues en détail. Nous préciserons ces points à l'aide des indications sur l'introduction du programme au moyen de la frappe des touches du clavier. C'est ce qu'illustre la figure 1 de ce chapitre 4. Auparavant, prenons le temps d'examiner cette figure, ce qui nous permet de constater qu'elle se compose des parties suivantes:
adresses Ø200 à Ø248 incluses, programme principal (fig. 20, ch. 3)
adresses Ø249 à Ø258 incluses, sous-programme SHIFT (fig. 21a, ch. 3)
adresses Ø259 à Ø26E incluses, sous-programme ADD (fig. 21b, ch. 3)
adresses Ø26F à Ø281 incluses, sous-programme KEYDIS (fig. 21c, ch. 3)
adresses Ø282 à Ø28A incluses, sous-programme CLB1 (fig. 21d, ch. 3)
adresses Ø28B à Ø293 incluses, sous-programme CLB2 (fig. 21e, ch. 3)
adresses Ø294 à Ø29C incluses, sous-programme CLDISP (fig. 21f, ch. 3)
adresses Ø29D à Ø2A9 incluses, sous-programme STO2 (fig. 21g, ch. 3)
adresses Ø2AA à Ø2B6 incluses, sous-programme STO1 (fig. 21h, ch. 3)
adresses Ø2B7 à Ø2C3 incluses, sous-programme RESDIS (fig. 21i, ch. 3)

Préparation de la frappe

Revenons-en au problème des éléments non encore connus précisément et reportons-nous à l'adresse Ø206 où est stocké le code opération de l'instruction JSR, grâce à laquelle le microprocesseur sautera à un sous-programme. Nous savons quel est celui-ci, mais au moment où nous ébauchons le programme de frappe des touches nous ne connaissons pas encore l'adresse de début du sous-programme, parce que nous ne savons pas quel est le nombre d'emplacements mémoire qui devront lui être réservés (le nombre d'adresses nécessaires). Ce qui nous amène à la formulation de deux règles générales très pratiques:

1. *Il est très instamment recommandé, après avoir procédé à l'établissement de l'ordinogramme détaillé, de faire précéder l'introduction physique du programme, au moyen du jeu de touches du clavier, par une rédaction sur papier de la séquence de frappe des touches!*

2. *Pour les opérandes et les octets d'offset non encore connus, il faut réserver un nombre suffisamment important d'emplacements mémoire, lesquels seront chargés de leur contenu ultérieurement.*

A cela s'ajoute une troisième règle, complément de la règle 2:

> *3. Les emplacements mémoire dans lesquels est inscrit un programme doivent être d'un seul tenant. Dans le cas d'un programme principal, comportant un ou plusieurs sous-programmes la même observation vaut pour ces derniers.*

Cette règle résulte du fait que, après l'exécution d'une instruction, le compteur ordinal (PC) contient automatiquement l'adresse de l'emplacement mémoire suivant. Et, lorsque l'emplacement concerné ne contient aucune instruction attendant d'être exécuté, le Junior Computer est condamné à l'inactivité. Et, naturellement, il ne peut jamais en être question.
Il arrive parfois qu'un emplacement mémoire reste vide, quand, par exemple, à l'occasion de l'introduction d'un programme en mode données au moyen du clavier (c'est-à-dire, lorsque des deux touches AD et DA, DA est la dernière pressée), la touche + est pressée deux fois consécutivement par erreur.
Ces emplacements mémoire vides peuvent d'ailleurs être chargés, après coup, avec l'octet EA qui est le code de l'instruction NOP (No OPeration). Cette instruction muette est susceptible d'être intercalée en un certain nombre d'emplacements mémoire (au stade de la conception sur papier de la séquence de frappe des touches) pour éliminer les conséquences fâcheuses résultant de l'oubli d'instructions et/ou d'opérandes, c'est-à-dire, d'octets. Car, rien n'est vraiment plus fastidieux que de devoir retaper entièrement un programme, par suite d'un oubli.
Nous avons déjà abordé cette question des erreurs; elles peuvent apparaître au moment de l'introduction du programme à l'aide du clavier, mais aussi au stade de la programmation, alors que l'on rédige sur papier la séquence de frappe. Il est donc important que l'on ait la faculté d'accéder à une adresse déterminée pour s'assurer de son contenu, ou encore de pouvoir réviser la totalité du programme après son introduction. En outre, il est indispensable de pouvoir faire des corrections.
C'est évidemment parfaitement possible.
Mais, auparavant, retenons encore ceci: *Dès que la tension d'alimentation cesse d'être appliquée au Junior Computer, tous les programmes introduits par le jeu de touches du clavier, depuis sa dernière mise sous tension, sont perdus!* C'est donc une erreur que d'imaginer, tandis que l'ordinateur reste sous tension, que l'on aura tout le temps, par la suite, d'éliminer une erreur du programme introduit.

Contrôle du contenu des emplacements mémoire

Abordons maintenant l'examen visuel des emplacements mémoire et la correction éventuelle de leur contenu. Les touches AD, DA et + jouent un rôle capital en la circonstance. Si l'on désire visualiser l'adresse d'un emplacement mémoire, on presse les touches dans l'ordre suivant:
AD X X X X
Les symboles XXXX sont, en fait, remplacés par l'adresse de l'emplace-

ment mémoire dont on veut connaître ou vérifier le contenu. Cette adresse est visualisée par les quatre afficheurs les plus à gauche; quant à l'octet contenu par l'emplacement mémoire, il apparaît, en notation hexadécimale, sur les deux afficheurs les plus à droite.
A supposer que l'on veuille vérifier ou connaître le contenu de l'emplacement mémoire situé à l'adresse suivante, il suffit de presser la touche +
Les afficheurs d'adresse font apparaître XXXX+ØØØ1, et les afficheurs de donnée visualisent la donnée présente en cet instant à l'adresse affichée.
La modification du contenu d'un emplacement mémoire exige que l'on affiche d'abord l'adresse, et donc, le contenu de celle-ci. Ensuite, on presse DA, puis les touches correspondant à l'octet en notation hexadécimale, et ces deux caractères sont affichés par les deux afficheurs de donnée. Si le contenu de l'emplacement mémoire suivant doit être modifié lui aussi, on presse la touche + , exactement comme on le fait pour l'introduction d'un programme au clavier. Grâce à ces manipulations, il est possible de vérifier si le programme de frappe des touches rédigé sur papier a été correctement tapé, et l'on peut y apporter d'éventuelles corrections.
Reprenons maintenant la question des octets d'adresses et de déplacement non encore connus au début de la rédaction sur papier de la séquence de frappe des touches. On ne sait pas encore vers quelle adresse le microprocesseur sera branché contitionnellement ou inconditionnellement, mais l'on connaît très bien la section de programme concernée, lorsque le branchement aura été réalisé. En fait, ces sections de programme sont précédées par ce que les informaticiens appellent une *étiquette* (label). Si vous examinez le programme de la figure 1, vous constatez qu'elles sont présentes dans la moitié droite de la page (le *champ commentaire*, qui explique ce qui se passe en un instant donné), encadrées par un rectangle. Les sous-programmes ont également chacun leur étiquette. Lors de l'établissement de la séquence de frappe, il est possible d'indiquer provisoirement l'étiquette de la section du programme vers laquelle le branchement aura lieu, à la place des octets d'adresses. Lorsque, ultérieurement, l'adresse correspondant au label sera connue, on la substituera à celui-ci.
Se pose alors la question de savoir à partir de quel moment opérandes et déplacements sont connus? Souvenons-nous que les déplacements peuvent être calculés à l'aide du Junior Computer; nous y reviendrons plus tard. Quant aux octets d'adresses, ils seront déterminés lorsque le nombre des emplacements mémoire nécessaires au stockage du programme vers lequel sera branché le microprocesseur sera précisé. Donc, lorsque l'évaluation des octets sera achevée.
Il est nécessaire non seulement que toute section de programme contienne une série d'emplacements mémoire d'un seul tenant, mais aussi que les sections ne se chevauchent pas. S'il arrive que, par erreur, l'on utilise la même adresse pour différents objectifs (autrement dit, pour différents programmes), seule la dernière introduction est validée. Si vous n'y prêtez attention, un sous-programme peut être stocké à des adresses nécessaires au programme principal.
D'ailleurs, il n'est pas impératif que le programme principal et des sous-programmes soient stockés de telle manière que l'adresse d'interruption de l'un soit suivie immédiatement par l'adresse de début de l'autre, comme c'est le cas de la figure 1. Il est bon qu'entre le programme principal et le

key				address	data	comments
RST				xxxx	xx	
AD				xxxx	xx	
Ø	2	Ø	Ø	Ø2ØØ	xx	
DA		2	Ø	Ø2ØØ	2Ø	JSR- CLEAR1
+		9	4	Ø2Ø1	94	ADL de CLDISP
+		Ø	2	Ø2Ø2	Ø2	ADH de CLDISP
+		2	Ø	Ø2Ø3	2Ø	JSR-
+		8	2	Ø2Ø4	82	ADL de CLB1
+		Ø	2	Ø2Ø5	Ø2	ADH de CLB1
+		2	Ø	Ø2Ø6	2Ø	JSR-
+		8	B	Ø2Ø7	8B	ADL de CLB2
+		Ø	2	Ø2Ø8	Ø2	ADH de CLB2
+		2	Ø	Ø2Ø9	2Ø	JSR- FIRST
+		6	F	Ø2ØA	6F	ADL de KEYDIS
+		Ø	2	Ø2ØB	Ø2	ADH de KEYDIS
+		C	9	Ø2ØC	C9	CMP#
+		1	Ø	Ø2ØD	1Ø	avec 1Ø
+		F	Ø	Ø2ØE	FØ	BEQ
+		F	Ø	Ø2ØF	FØ	CLEAR1 est à FØ emplacements en arrière.
+		C	9	Ø21Ø	C9	CMP#
+		1	2	Ø211	12	avec 12
+		F	Ø	Ø212	FØ	BEQ
+		Ø	6	Ø213	Ø6	(déplacement) PLUS est Ø6 emplacements plus loin
+		2	Ø	Ø214	2Ø	JSR-
+		4	9	Ø215	49	ADL de SHIFT
+		Ø	2	Ø216	Ø2	ADH de SHIFT
+		4	C	Ø217	4C	JMP- (saut inconditionnel)
+		Ø	9	Ø218	Ø9	ADL de FIRST
+		Ø	2	Ø219	Ø2	ADH de FIRST
+		2	Ø	Ø21A	2Ø	JSR- PLUS
+		A	A	Ø21B	AA	ADL de STO1
+		Ø	2	Ø21C	Ø2	ADH de STO1
+		2	Ø	Ø21D	2Ø	JSR-
+		9	4	Ø21E	94	ADL de CLDISP
+		Ø	2	Ø21F	Ø2	ADH de CLDISP
+		2	Ø	Ø22Ø	2Ø	JSR- SECOND
+		6	F	Ø221	6F	ADL de KEYDIS
+		Ø	2	Ø222	Ø2	ADH de KEYDIS
+		C	9	Ø223	C9	CMP#
+		1	Ø	Ø224	1Ø	avec 1Ø
+		F	Ø	Ø225	FØ	BEQ
+		Ø	A	Ø226	ØA	(déplacement) CLEAR2 est à ØA emplacements plus loin
+		C	9	Ø227	C9	CMP#
+		1	1	Ø228	11	avec 11
+		F	Ø	Ø229	FØ	BEQ
+		Ø	C	Ø22A	ØC	(déplacement) EQUAL est à ØC emplacements plus loin
+		2	Ø	Ø22B	2Ø	JSR-
+		4	9	Ø22C	49	ADL de SHIFT
+		Ø	2	Ø22D	Ø2	ADH de SHIFT
+		4	C	Ø22E	4C	JMP-
+		2	Ø	Ø22F	2Ø	ADL de SECOND
+		Ø	2	Ø23Ø	Ø2	ADH de SECOND

+	2	Ø	Ø231	2Ø	JSR- CLEAR2
+	9	4	Ø232	94	ADL de CLDISP
+	Ø	2	Ø233	Ø2	ADH de CLDISP
+	4	C	Ø234	4C	JMP-
+	2	Ø	Ø235	2Ø	ADL de SECOND
+	Ø	2	Ø236	Ø2	ADH de SECOND
+	2	Ø	Ø237	2Ø	JSR- EQUAL
+	9	D	Ø238	9D	ADL de STO2
+	Ø	2	Ø239	Ø2	ADH de STO2
+	2	Ø	Ø23A	2Ø	JSR-
+	5	9	Ø23B	59	ADL de ADD
+	Ø	2	Ø23C	Ø2	ADH de ADD
+	2	Ø	Ø23D	2Ø	JSR-
+	B	7	Ø23E	B7	ADL de RESDIS
+	Ø	2	Ø23F	Ø2	ADH de RESDIS
+	2	Ø	Ø24Ø	2Ø	JSR-
+	8	2	Ø241	82	ADL de CLB1
+	Ø	2	Ø242	Ø2	ADH de CLB1
+	2	Ø	Ø243	2Ø	JSR-
+	8	B	Ø244	8B	ADL de CLB2
+	Ø	2	Ø245	Ø2	ADH de CLB2
+	4	C	Ø246	4C	JMP-
+	Ø	9	Ø247	Ø9	ADL de FIRST
+	Ø	2	Ø248	Ø2	ADH de FIRST (dernière ligne du programme principal
+	A	Ø	Ø249	AØ	LD# sous-programme SHIFT
+	Ø	4	Ø24A	Ø4	Ø4 dans le registre index y
+	Ø	6	Ø24B	Ø6	ASLZ SHIFT1
+	F	9	Ø24C	F9	INH à l'adresse ØØF9
+	2	6	Ø24D	26	ROLZ
+	F	A	Ø24E	FA	POINTL à l'adresse ØØFA
+	2	6	Ø24F	26	ROLZ
+	F	B	Ø25Ø	FB	POINTH à l'adresse ØØFB
+	8	8	Ø251	88	décrémenter de 1 le contenu de Y
+	D	Ø	Ø252	DØ	BNE
+	F	7	Ø253	F7	SHIFT1 est à F7 emplacements en arrière
+	Ø	5	Ø254	Ø5	ORAZ
+	F	9	Ø255	F9	fonction OR bit par bit avec INH
+	8	5	Ø256	85	STAZ
+	F	9	Ø257	F9	contenu de l'accu vers INH (adr. ØØF9)
+	6	Ø	Ø258	6Ø	RTS retour au programme principal
+	F	8	Ø259	F8	SED calcul décimal sous-programme ADD
+	1	8	Ø25A	18	CLC
+	A	5	Ø25B	A5	LDAZ
+	Ø	Ø	Ø25C	ØØ	ADL de B1Ø (adr. ØØØØ); B1Ø dans l'accu
+	6	5	Ø25D	65	ADCZ
+	Ø	3	Ø25E	Ø3	ADL de B2Ø (adr. ØØØ3); accu = B1Ø + B2Ø
+	8	5	Ø25F	85	STAZ
+	Ø	6	Ø26Ø	Ø6	ADL de RØ; accu → RØ
+	A	5	Ø261	A5	LDAZ
+	Ø	1	Ø262	Ø1	ADL de B11 (adr. ØØØ1); B11 dans l'accu
+	6	5	Ø263	65	ADCZ
+	Ø	4	Ø264	Ø4	ADL de B21 (adr. ØØØ4); accu = B11 + B21

+	8	5	Ø265	85	STAZ
+	Ø	7	Ø266	Ø7	ADL de R1 (adr. ØØØ7); accu → R1
+	A	5	Ø267	A5	LDAZ
+	Ø	2	Ø268	Ø2	ADL de B12 (adr. ØØØ2); B12 dans l'accu
+	6	5	Ø269	65	ADCZ
+	Ø	5	Ø26A	Ø5	ADL de B22 (adr. ØØØ5), accu = B12 + B22
+	8	5	Ø26B	85	STAZ
+	Ø	8	Ø26C	Ø8	ADL de R2 (adr. ØØØ8); accu → R2
+	D	8	Ø26D	D8	CLD retour au mode de calcul binaire
+	6	Ø	Ø26E	6Ø	RTS retout au programme principal
+	2	Ø	Ø26F	2Ø	JSR- sous-programme KEYDIS
+	8	E	Ø270	8E	ADL de SCANDS } dans le moniteur (1D8E)
+	1	D	Ø271	1D	ADH de SCANDS }
+	D	Ø	Ø272	DØ	BNE
+	F	B	Ø273	FB	(déplacement) KEYDIS est à FB emplacements en arrière
+	2	Ø	Ø274	2Ø	JSR- KD
+	8	E	Ø275	8E	ADL de SCANDS } dans le moniteur (1D8E)
+	1	D	Ø276	1D	ADH de SCANDS }
+	F	Ø	Ø277	FØ	BEQ
+	F	B	Ø278	FB	(déplacement) KD est à FB emplacements en arrière
+	2	Ø	Ø279	2Ø	JSR-
+	8	E	Ø27A	8E	ADL de SCANDS } dans le moniteur (1D8E)
+	1	D	Ø27B	1D	ADH de SCANDS }
+	F	Ø	Ø27C	FØ	BEQ
+	F	6	Ø27D	F6	(déplacement) KD est à F6 emplacements en arrière
+	2	Ø	Ø27E	2Ø	JSR-
+	F	9	Ø27F	F9	ADL de GETKEY } dans le moniteur (1DF9)
+	1	D	Ø28Ø	1D	ADH de GETKEY }
+	6	Ø	Ø281	6Ø	RTS retour au programme principal
+	A	9	Ø282	A9	LDA# sous-programme CLB1
+	Ø	Ø	Ø283	ØØ	ØØ → accumulateur
+	8	5	Ø284	85	STAZ
+	Ø	Ø	Ø285	ØØ	accu → B1Ø (= ØØ)
+	8	5	Ø286	85	STAZ
+	Ø	1	Ø287	Ø1	ØØ → B11
+	8	5	Ø288	85	STAZ
+	Ø	2	Ø289	Ø2	ØØ → B12
+	6	Ø	Ø28A	6Ø	RTS retour au programme principal
+	A	9	Ø28B	A9	LDA# sous-programme CLB2
+	Ø	Ø	Ø28C	ØØ	ØØ → accu
+	8	5	Ø28D	85	STAZ
+	Ø	3	Ø28E	Ø3	ØØ → B2Ø
+	8	5	Ø28F	85	STAZ
+	Ø	4	Ø29Ø	Ø4	ØØ → B21
+	8	5	Ø291	85	STAZ
+	Ø	5	Ø292	Ø5	ØØ → B22
+	6	Ø	Ø293	6Ø	RTS retour au programme principal
+	A	9	Ø294	A9	LDA# sous-programme CLDISP
+	Ø	Ø	Ø295	ØØ	ØØ → accu
+	8	5	Ø296	85	STAZ
+	F	9	Ø297	F9	ØØ → INH (adr. ØØF9)

+	8	5	Ø298	85	STAZ
+	F	A	Ø299	FA	ØØ → POINTL (adr. ØØFA)
+	8	5	Ø29A	85	STAZ
+	F	B	Ø29B	FB	ØØ → POINTH (adr. ØØFB)
+	6	Ø	Ø29C	6Ø	RTS retour au programme principal
+	A	5	Ø29D	A5	LDAZ sous-programme STO2
+	F	9	Ø29E	F9	INH (adr. ØØF9) → accu
+	8	5	Ø29F	85	STAZ
+	Ø	3	Ø2AØ	Ø3	accu (= INH) → B2Ø (adr. ØØØ3)
+	A	5	Ø2A1	A5	LDAZ
+	F	A	Ø2A2	FA	POINTL (adr. ØØFA) → accu
+	8	5	Ø2A3	85	STAZ
+	Ø	4	Ø2A4	Ø4	accu (= POINTL) → B21 (adr. ØØØ4)
+	A	5	Ø2A5	A5	LDAZ
+	F	B	Ø2A6	FB	POINTH (adr. ØØFB) → accu
+	8	5	Ø2A7	85	STAZ
+	Ø	5	Ø2A8	Ø5	accu (= POINTH) → B22 (adr. ØØØ5)
+	6	Ø	Ø2A9	6Ø	RTS retour au programme principal
+	A	5	Ø2AA	A5	LDAZ sous-programme STO1
+	F	9	Ø2AB	F9	INH (adr. ØØØF9) → accu
+	8	5	Ø2AC	85	STAZ
+	Ø	Ø	Ø2AD	ØØ	accu (= INH) → B1Ø (adr. ØØØØ)
+	A	5	Ø2AE	A5	LDAZ
+	F	A	Ø2AF	FA	POINTL (adr. ØØFA) → accu
+	8	5	Ø2BØ	85	STAZ
+	Ø	1	Ø2B1	Ø1	accu (= POINTL) → B11 (adr. ØØØ1)
+	A	5	Ø2B2	A5	LDAZ
+	F	B	Ø2B3	FB	POINTH (adr. ØØFB) → accu
+	8	5	Ø2B4	85	STAZ
+	Ø	2	Ø2B5	Ø2	accu (= POINTH) → B12 (adr. ØØØ2)
+	6	Ø	Ø2B6	6Ø	RTS retour au programme principal
+	A	5	Ø2B7	A5	LDAZ sous-programme RESDIS
+	Ø	6	Ø2B8	Ø6	RØ (adr. ØØØ6) → accu
+	8	5	Ø2B9	85	STAZ
+	F	9	Ø2BA	F9	accu (= RØ) → INH (adr. ØØF9)
+	A	5	Ø2BB	A5	LDAZ
+	Ø	7	Ø2BC	Ø7	R1 (adr. ØØØ7) → accu
+	8	5	Ø2BD	85	STAZ
+	F	A	Ø2BE	FA	accu (= R1) → POINTL (adr. ØØFA)
+	A	5	Ø2BF	A5	LDAZ
+	Ø	8	Ø2CØ	Ø8	R2 (adr. ØØØ8) → accu
+	8	5	Ø2C1	85	STAZ
+	F	B	Ø2C2	FB	accu (= R2) → POINTH (adr. ØØFB)
+	6	Ø	Ø2C3	6Ø	RTS retour au programme principal

Figure 1. Sequence d'introduction du programme d'addition décimale du chapitre 3, à l'aide du jeu de touches du clavier du Junior Computer. Le programme se compose de 196 octets et occupe donc 196 emplacements mémoire. Son ordinogramme détaillé est présenté par la figure 20 du chapitre 3 et celui des neuf sous-programmes par les figures 21a à 21i du même chapitre.

premier sous-programme, ainsi qu'entre les sous-programmes eux-mêmes, des emplacements mémoire vides (contenant l'octet EA, code opération de NOP, en réalité) soient ménagés. Bien entendu, on ne peut perdre de vue la nécessaire économie des emplacements mémoire, et il faut donc réduire le nombre des emplacements non utilisés au minimum compatible avec une gestion avisée de la capacité mémoire. C'est surtout le cas lorsque le programme est important. Mais, avec un programme relativement court, on peut se permettre d'être plus généreux. Quand, par exemple, un programme occupe 50 emplacements mémoire, rien ne s'oppose à ce que l'on prévoie 100 emplacements mémoire pour le sous-programme, à partir de l'adresse de début du programme principal. On peut alors introduire immédiatement les octets d'adresses et il n'est plus nécessaire de le faire par la suite.
De quel nombre d'emplacements mémoire dispose-t-on?

4. La RAM 1K mise à la disposition de l'utilisateur dans la version standard du Junior Computer est composée de quatre pages, ayant chacune une capacité de 256 octets dont les adresses vont de ØØØØ . . . Ø3FF:

page ØØ : ØØØØ . . . ØØFF
page Ø1 : Ø1ØØ . . . Ø1FF
page Ø2 : Ø2ØØ . . . Ø2FF
page Ø3 : Ø3ØØ . . . Ø3FF

Mais il est bon de préciser qu'il existe quelques contraintes pour la page zéro, car un certain nombre d'emplacements y sont réservés à des données résultant de manipulations du programme moniteur. Ce sont 31 emplacements mémoire dont les adresses vont de ØØE1 à ØØFF. Certaines d'entre elles ne nous sont pas inconnues, comme, par exemple, ØØF9 (étiquette INH), ØØFA (POINTL) et ØØFB (POINTH). En outre, les emplacements ØØEF . . . ØØF5 sont réservés au stockage du contenu de tous les registres internes du 6502 (routine SAVE). Le second livre consacré au Junior Computer vous permettra de faire connaissance avec les autres emplacements mémoire de la page zéro réservés au programme moniteur.
L'utilisateur doit réserver lui-même d'autres emplacements de la page zéro s'il fait usage de l'adressage en cette page.
La page Ø1 peut servir de pile (stack).
D'autres emplacements mémoire ne peuvent recevoir d'inscriptions.

5. L' EPROM 1K du Junior Computer contient le programme moniteur qui occupe les adresses 1CØØ à 1 FFF, réparties sur les pages 1C, 1D, 1E et 1F. Ces emplacements mémoire ne sont accessibles à l'utilisateur que dans le mesure où il fait appel à des routines moniteur déterminées.

C'est ce que nous montre également la figure 1 où l'on voit qu'il est fait usage du sous-programme KEYDIS (adresses Ø26F à Ø281) lequel fait appel aux routines moniteur SCANDS et GETKEY.

Mais nous n'y sommes pas encore, car:

6. La RAM $\frac{1}{8}$K du PIA est à la disposition de l'utilisateur du Junior Computer pour y charger ses propres programmes. Elle contient les adresses 1AØØ à 1A7F réparties sur la première moitié de la page 1A.

Exactement comme ce fut le cas pour la page zéro de la RAM standard, il existe aussi certaines contraintes, car les quatre emplacements mémoire 1A7A, 1A7B, 1A7E et 1A7F sont réservés aux vecteurs-saut NMI et IRQ. L'utilisateur ne peut les inclure dans son programme que s'il prévoit de ne pas faire usage des instructions NMI et IRQ (voir le paragraphe "Mise en service", ci-après).
Une partie des emplacements mémoire de la seconde moitié de la page 1A (adresses 1A8Ø à 1AFB) est nécessaire au fonctionnement du PIA. Nous reviendrons sur ce sujet dans le second livre. Il faut donc éviter (provisoirement) de s'en servir lors de la programmation.

Déplacement: L'octet de branchement

Pour le calcul du déplacement d'une instruction de branchement conditionnel, nous pouvons nous servir de la routine moniteur BRANCH dont l'adresse de début est 1FD5, ainsi que nous l'avons vu au chapitre 3. Après que le programme ait été lancé par une pression sur la touche GO , nous tapons d'abord l'octet droit ADL de l'adresse du code opératoire de l'instruction de branchement conditionnel, puis l'octet de l'adresse du branchement. Calculons tous les déplacements de la figure 1:

RST	AD			XXXX	XX	
1	F	D	5	1FD5	D8	
GO				ØØØØ	ØØ	
Ø	E	Ø	Ø	ØEØØ	FØ	FØ à l'adresse Ø2ØF
1	2	1	A	121A	Ø6	Ø6 à l'adresse Ø213
2	5	3	1	2531	ØA	ØA à l'adresse Ø226
2	9	3	7	2937	ØC	ØC à l'adresse Ø22A
5	2	4	B	524B	F7	F7 à l'adresse Ø253
7	2	6	F	726F	FB	FB à l'adresse Ø273
7	7	7	4	7774	FB	FB à l'adresse Ø278
7	C	7	4	7C74	F6	F6 à l'adresse Ø27D
RST						

Entre deux calculs, il est possible de remettre l'affichage à ØØØØØØ en pressant l'une des touches de commande.
D'autre part, il n'est pas indispensable que la détermination du déplacement se fasse par l'intermédiaire du programme moniteur; il suffit de compter le nombre de pas nécessaires pour que soit atteinte l'adresse de branchement. Chaque incrémentation ou décrémentation de l'adresse permet d'avancer ou de reculer d'un pas. *Le point de départ du décompte des pas n'est pas l'adresse comportant le code opération de l'instruction de*

branchement, mais l'adresse de l'emplacement situé deux pas plus loin, donc l'adresse de l'emplacement mémoire faisant suite à celui où est stocké l'octet de déplacement. Les pas arrière sont négatifs. Grâce au complément à deux du nombre, on obtient ensuite l'octet en notation hexadécimale. La figure 5 du chapitre 2 nous montre un extrait de la notation hexadécimale des nombres jusqu'à –40.

Mise en service

La mise en exploitation du Junior Computer a déjà été abordée dans le cours du chapitre 3. Néanmoins, nous y revenons:

```
RST        AD        XXXX XX
 1   A     7   A     1A7A XX
 DA        Ø   Ø     1A7A ØØ
 +         1   C     1A7B 1C
 +                   1A7C XX
 +                   1A7D XX
 +         Ø   Ø     1A7E ØØ
 +         1   C     1A7F 1C
```

En fait, nous avons procédé à l'écriture de ØØ à l'adresse 1A7E et de 1C à l'adresse 1A7F. Ces emplacements mémoire (de la page 1A de la RAM du PIA) contiennent l'adresse effective (1CØØ) du programme moniteur vers laquelle se fera le branchement dès qu'apparaîtra une instruction BRK, qui est la version "software" (logiciel) de l'instruction "hardware" (matériel) IRQ. La dernière partie du chapitre 3, illustrée par la figure 31, est consacrée à cette question. Les emplacements mémoire 1A7A et 1A7B contiennent également l'adresse effective 1CØØ. L'apparition d'une instruction NMI provoquera le branchement du microordinateur vers cette adresse. Dans la version standard du Junior Computer, c'est le cas lorsque la touche ST est pressée.

La pression exercée sur la touche RST lors du lancement a pour conséquence un branchement vers les emplacements mémoire 1FFC+1FFD où est stocké le vecteur-reset 1C1D, qui pointe vers la partie du programme moniteur chargée de l'interrogation des touches et de l'affichage des adresses ainsi que des données tapées grâce au clavier.

La procédure de mise en service indiquée peut être parfois simplifiée. Si l'on renonce à l'utilisation de l'instruction BRK, il n'est pas nécessaire de charger les emplacements mémoire 1A7E et 1A7F; en écartant l'emploi de la touche ST et de la programmation pas à pas, les emplacements mémoire 1A7A et 1A7B ne devront pas l'être non plus. Si l'on ne se sert ni de l'une ni de l'autre, une pression sur RST peut suffire. Dans les trois cas, le vecteur-saut pointe vers le programme moniteur. La pression sur la touche RST permet d'atteindre directement l'adresse 1C1D du programme moniteur; une instruction NMI ou IRQ (=BRK) permet d'atteindre l'adresse 1C1D après qu'une section de programme, dont l'adresse de début est 1CØØ, ait été parcourue afin que soit assurée la sauvegarde du contenu de tous les registres du 6502. Bien entendu, tout de suite après la mise en route du Junior Computer, il n'y a rien à sauvegarder, d'où le branchement direct vers 1C1D après une pression sur la touche RST .

Les préparatifs étant terminés et le programme ayant été correctement introduit au moyen du jeu de touches, nous pouvons l'exécuter:

AD	Ø	2	Ø	Ø		(introduction de l'adresse de début)
GO						(lancement du programme)
						Affichage:
	2	4	5	6		ØØ2456
+						ØØØØØØ
	4	1	3	2		ØØ4132
DA						ØØ6588 (la touche DA est aussi la touche =)
AD						ØØØØØØ (AD = CLEAR = mise à zéro affichage)
1	9	8	5	3	1	198531
+						ØØØØØØ
8	3	2	7	Ø	2	832702
DA						Ø31233
AD						ØØØØØØ

Pour la dernière opération d'addition, le résultat excède 999999 et l'on a un cas de dépassement (overflow). Seuls les 6 chiffres les plus à droite du résultat sont affichés.

Jeux de dés avec le Junior Computer

L'exemple de programme qui va suivre a surtout une valeur éducative. Les dés sont bon marché, et les jeux de dés électroniques sont facilement réalisables.

Une pression sur la touche + équivaut au jet des dés. Dès que cette touche est relâchée, les dés cessent de rouler. Leur face supérieure indique un nombre compris entre 1 et 6. La valeur du point est déterminée par un compteur logiciel susceptible de compter séquentiellement, aussi bien de 1 à 7 que de 1 à 6 inclus. Le comptage débute dès que la touche + est pressée, et il cesse dès qu'elle est relâchée. L'état du compteur au moment où la touche est relâchée (Ø1, Ø2, Ø3, Ø4, Ø5, Ø6) est visualisé sur les deux afficheurs au centre de l'affichage (qui, normalement, indiquent l'octet d'adresse droit ADL). Les deux afficheurs les plus à gauche et les deux les plus à droite font apparaître "FF".

L'ordinogramme général du jeu de dés logiciel est donné en figure 2. Il est fait usage des routines moniteur SCANDS, AK et GETKEY que nous avons déjà rencontrées au chapitre 3.

Voyons ce que nous indique la figure 2. Tout d'abord, l'affichage fait apparaître FFFFFF. Ensuite, commence la section de programme étiquetée SCAN1. Elle débute par la routine moniteur SCANDS qui assure que le contenu des trois tampons d'affichage (voir la figure 14 du chapitre 3) soit affiché; en outre, elle interroge le jeu de touches afin de détecter si l'une d'entre elles a été pressée. Cette mission est confiée à la routine AK qui fait immédiatement suite à la routine SCANDS. Tant qu'aucune touche n'est pressée, le microprocesseur revient à la routine SCAN1 (boucle d'attente). Si, par contre, une touche est pressée, toute la procédure associée au sous-programme SCANDS est répétée. Le chapitre 3 nous a appris que c'est ainsi que les rebonds des touches sont ignorés par le programme. Puis, la routine GETKEY détermine quelle est la touche

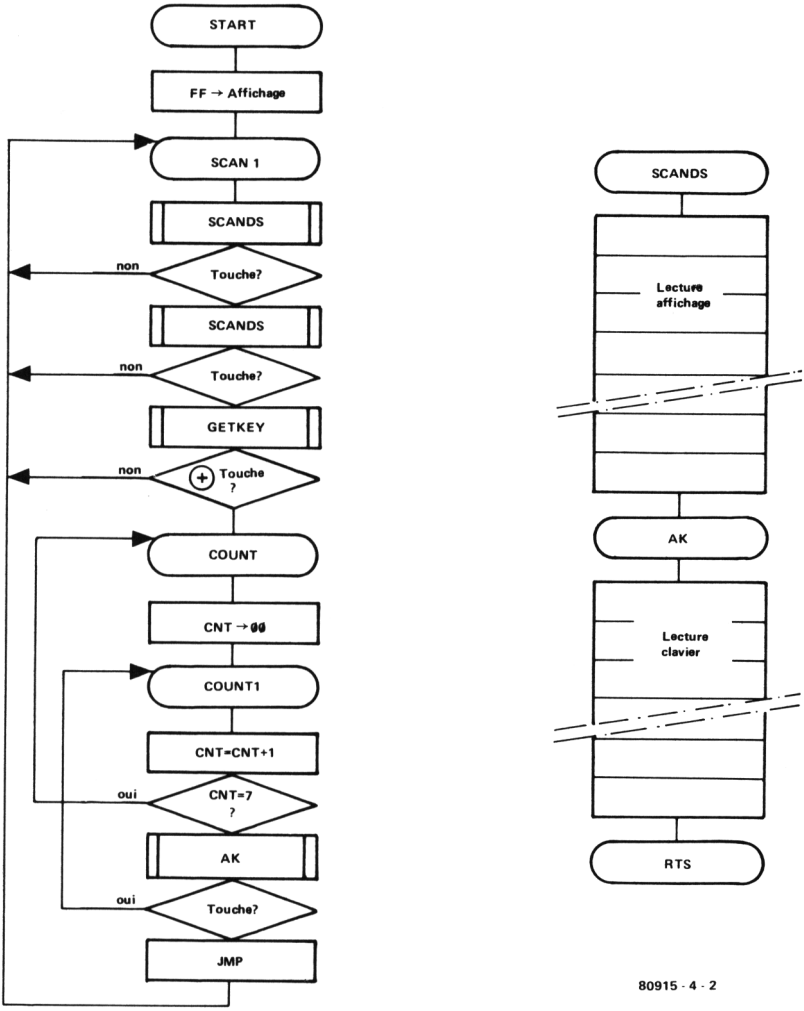


Figure 2. Ordinogramme général du jeu de dés logiciel. Il fait appel à trois routines différentes du programme moniteur.

pressée. Après décodage de la touche, le programme demande s'il s'agit de la touche + , car seule celle-ci sera acceptée, toutes les autres étant ignorées.
Dès qu'il s'agit bien de la touche + , débute le comptage séquentiel proprement dit (COUNT). L'ordinogramme détaillé de la figure 3 montre que le nombre ØØ est chargé dans le tampon d'affichage POINTL, par l'intermédiaire de l'accumulateur, pour les deux afficheurs centraux. Ensuite, (COUNT1), le contenu de POINTL (=CNT) est incrémenté de 1 et

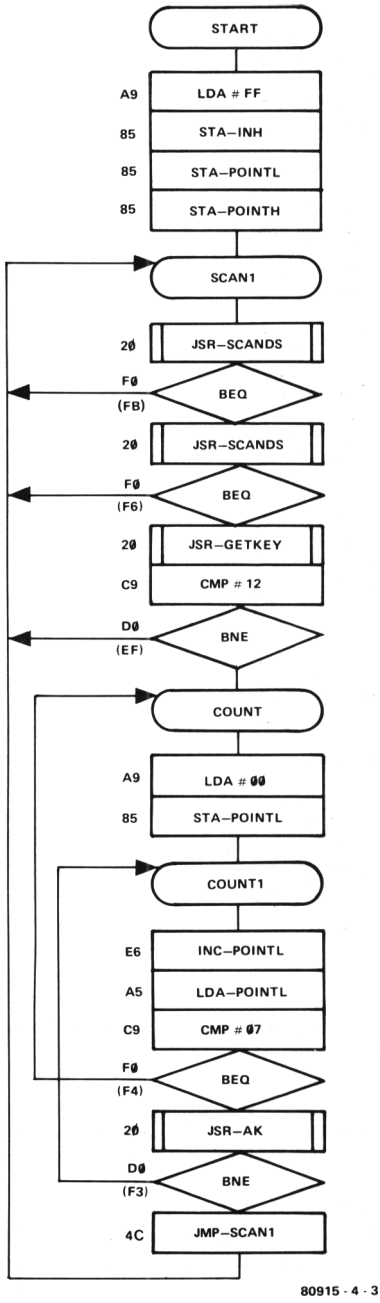


Figure 3. Ordinogramme détaillé du jeu de dés.

transféré dans l'accumulateur. Alors, le microprocesseur vérifie que l'état du compteur CNT n'est pas égal à Ø7, ce qui ne peut être le cas au premier contrôle; après que la boucle COUNT1 ait été parcourue sept fois, c'est ce qui se produirait. Mais à partir du moment où l'état du compteur menace de passer à Ø7, le programme revient à COUNT et le contenu est remis à ØØ. L'état du compteur n'étant pas égal à Ø7, le microprocesseur revient à la routine AK qui contrôle si la touche + continue à être pressée. Dans l'affirmative, le comptage se poursuit (branchement vers COUNT1); dans la négative, le comptage cesse. Le microprocesseur revient à la routine SCAN1, par l'intermédiaire d'une instruction JMP, et, tout de suite après, la routine SCANDS fait apparaître le contenu du compteur sur les afficheurs. Puis, tout le processus recommence.

La figure 3 montre le processus grâce auquel l'état du compteur est vérifié. Celui-ci est transféré dans l'accumulateur, puis il est comparé à Ø7, grâce à une instruction CMP.

Cette figure nous permet de constater que la section de programme COUNT1 comporte un nombre important d'instructions. L'exécution de chacune d'entre elles exige quelques microsecondes (voir les indications à ce sujet dans l'annexe 2). Cela signifie que le compteur doit fonctionner suffisamment vite comparativement au temps pendant lequel la touche + est pressée, ce dernier point étant indispensable pour que l'état du compteur ne puisse être influencé par les manipulations d'un tricheur.

La figure 4 est l'illustration de la séquence de frappe du jeu de touches pour l'introduction du programme. L'adresse de début est également Ø2ØØ et la séquence s'accompagne d'un nombre important de commentaires. Le lancement du programme débute par:

AD Ø 2 Ø Ø GO

Ensuite, il n'y a plus qu'à presser la touche + autant de fois qu'on le voudra, dans l'espoir d'obtenir le nombre de points le plus élevé possible.

	toetsen			adres	data		
RST		AD		xxxx	xx		
Ø	2	Ø	Ø	Ø2ØØ	xx		
DA		A	9	Ø2ØØ	A9	LDA IMM	START
+		F	F	Ø2Ø1	FF	FF → accu	
+		8	5	Ø2Ø2	85	STA Z	
+		F	9	Ø2Ø3	F9	FF → INH (ØØF9)	
+		8	5	Ø2Ø4	85	STA Z	
+		F	A	Ø2Ø5	FA	FF → POINTL (ØØFA)	
+		8	5	Ø2Ø6	85	STA Z	
+		F	B	Ø2Ø7	FB	FF → POINTH (ØØFB)	
+		2	Ø	Ø2Ø8	2Ø	JSR-	SCAN1
+		8	E	Ø2Ø9	8E	ADL } de SCANDS dans le moniteur	
+		1	D	Ø2ØA	1D	ADH } (adr.1D8E)	
+		F	Ø	Ø2ØB	FØ	BEQ	
+		F	B	Ø2ØC	FB	(déplacement) SCAN1 est à FB emplacements en arrière	
+		2	Ø	Ø2ØD	2Ø	JSR-	

+		8	E	Ø20E	8E	ADL } de SCANDS dans le moniteur
+		1	D	Ø20F	1D	ADH } (adr. 1D8E)
+		F	Ø	Ø210	FØ	BEQ
+		F	6	Ø211	F6	(déplacement) SCAN1 est à F6 emplacements en arrière
+		2	Ø	Ø212	2Ø	JSR-
+		F	9	Ø213	F9	ADL } de GETKEY dans le moniteur
+		1	D	Ø214	1D	ADH } (adr. 1DF9)
+		C	9	Ø215	C9	CMP IMM
+		1	2	Ø216	12	12 avec 12
+		D	Ø	Ø217	DØ	BNE
+		E	F	Ø218	EF	(déplacement) SCAN1 est à EF emplacements en arrière
+		A	9	Ø219	A9	LDA IMM COUNT
+		Ø	Ø	Ø21A	ØØ	ØØ → accu
+		8	5	Ø21B	85	STA Z
+		F	A	Ø21C	FA	ØØ → POINTL (ØØFA)
+		E	6	Ø21D	E6	INC Z COUNT1
+		F	A	Ø21E	FA	POINTL + 1 → POINTL
+		A	5	Ø21F	A5	LDA Z
+		F	A	Ø22Ø	FA	POINTL → accu
+		C	9	Ø221	C9	CMP IMM
+		Ø	7	Ø222	Ø7	avec Ø7
+		F	Ø	Ø223	FØ	BEQ
+		F	4	Ø224	F4	COUNT est à F4 emplacements en arrière
+		2	Ø	Ø225	2Ø	JSR-
+		B	1	Ø226	B1	ADL } de AK dans le moniteur (adr. 1DB1)
+		1	D	Ø227	1D	ADH }
+		D	Ø	Ø228	DØ	BNE
+		F	3	Ø229	F3	COUNT1 est à F3 emplacements en arrière
+		4	C	Ø22A	4C	JMP-
+		Ø	8	Ø22B	Ø8	ADL } van SCAN1
+		Ø	2	Ø22C	Ø2	ADH }
AD						
Ø	2	Ø	Ø	Ø2ØØ	A9	adresse de début de programme
GO						lancement du programme
+				FFØ4	FF	le sort en est jeté!
+				FFØ1	FF	
+				FFØ6	FF	
+				FFØ2	FF	

et ainsi de suite . . .

Figure 4. Séquence d'introduction du programme du jeu de dés à l'aide du clavier du Junior Computer.

Détermination de la longueur des instructions par le logiciel

Les instructions comportent un, deux ou trois octets. Un octet est réservé au code opération, les autres l'étant à l'opérande, qui, lui-même, peut ne pas être présent (pas d'octet) ou s'exprimer par un ou deux octets.

Le code opération est formé d'un octet, soit deux caractères hexadécimaux nécessitant chacun un quartet pour leur définition. A la fin de cet ouvrage, dans l'appendice 1, se trouve un tableau des 256 combinaisons possibles de deux caractères hexadécimaux accompagnées, à toutes fins utiles, du mnémonique correspondant et du mode d'adressage (celui-ci exclusivement pour les instructions justifiables de plusieurs modes d'adressage). L'appendice 1 présente également un tableau très condensé inspiré de celui de la figure 5. Pour chaque rangée de cette figure, le quartet gauche est identique. Les 256 éléments du tableau se composent de 29 codes opération d'instructions de 1 octet, 74 codes opération d'instructions de 3 octets, et de 105 emplacements vides.

Nous voulons écrire un programme capable de déterminer, lorsqu'on introduit un octet grâce au jeu de touches, s'il s'agit:

- du code opération d'une instruction de 1 octet ou
- du code opération d'une instruction de 2 octets ou
- du code opération d'une instruction de 3 octets ou, parfois,
- d'autre chose qu'un code opération

En la circonstance, il s'agit surtout d'un "amusement" éducatif, car le programme concerné est presque toujours inclus sous forme de sous-programme dans l'assembleur et l'éditeur d'un microordinateur (nous reviendrons sur ce sujet dans le second livre).

De ce programme, que l'on peut considérer comme une sorte de "calibre d'instruction", nous abordons d'abord le sous-programme LENACC dont l'ordinogramme détaillé est montré en figure 6. En réalité, il s'agit du "corps" du programme. Il débute après le rangement de l'octet tapé au clavier, dans l'accumulateur; à la fin, la longueur de l'instruction est chargée à l'emplacement mémoire BYTES. Nous examinerons ultérieurement le chargement de l'octet dans l'accumulateur et l'affichage du contenu de BYTES.

Pendant le programme, le registre X contient l'information relative à la longueur de l'instruction; si X=ØØ, cela signifie qu'il n'y a pas de code opération (instruction à octet nul), si X=Ø1, instruction d'un octet, si X=Ø2, la longueur de l'instruction est de deux octets, si X=Ø3, elle est de trois octets. A la fin du sous-programme LENACC, dans la section LENEND, le contenu de X est inscrit à l'emplacement mémoire BYTES.

Dans ce programme, le registre y joue également un rôle passif. En effet, quand un octet tapé ne semble pas faire partie des exceptions (dont nous allons bientôt reparler), la valeur du quartet droit est chargée dans y; elle est indiquée dans une colonne déterminée du tableau des codes opération de la figure 5. Ensuite, le registre Y joue le rôle d'index pour le chargemment de X grâce à l'adressage indexé Y absolu. Le registre X intervient dans la détermination de la longueur de l'instruction, car le contenu du "Tableau indicateur" y est rangé sous l'étiquette LENTBL. Plus précisément, il s'agit du contenu de y qui est celui de l'emplacement mémoire LENTBL. Pour chaque colonne (Ø . . . F), le tableau indicateur contient la longueur correspondant à chaque instruction de la colonne (par

Caractère hexadécimal gauche	Caractère hexadécimal droit							
	Ø	1	2	3	4	5	6	7
Ø	BRK (1)	ORA (IND,X) (2)				ORA Z (2)	ASL Z (2)	
1	BPL (2)	ORA (IND),Y (2)				ORA Z,X (2)	ASL Z,X (2)	
2	JSR (3)	AND (IND,X) (2)			BIT Z (2)	AND Z (2)	ROL Z (2)	
3	BMI (2)	AND (IND),Y (2)				AND Z,X (2)	ROL Z,X (2)	
4	RTI (1)	EOR (IND,X) (2)				EOR Z (2)	LSR Z (2)	
5	BVC (2)	EOR (IND),Y (2)				EOR Z,X (2)	LSR Z,X (2)	
6	RTS (1)	ADC (IND,X) (2)				ADC Z (2)	ROR Z (2)	
7	BVS (2)	ADC (IND),Y (2)				ADC Z,X (2)	ROR Z,X (2)	
8		STA (IND,X) (2)			STY Z (2)	STA Z (2)	STX Z (2)	
9	BCC (2)	STA (IND),Y (2)			STY Z,X (2)	STA Z,X (2)	STX Z,Y (2)	
A	LDY # (2)	LDA (IND,X) (2)	LDX # (2)		LDY Z (2)	LDA Z (2)	LDX Z (2)	
B	BCS (2)	LDA (IND),Y (2)			LDY Z,X (2)	LDA Z,X (2)	LDX Z,Y (2)	
C	CPY # (2)	CMP (IND,X) (2)			CPY Z (2)	CMP Z (2)	DEC Z (2)	
D	BNE (2)	CMP (IND),Y (2)				CMP Z,X (2)	DEC Z,X (2)	
E	CPX # (2)	SBC (IND,X) (2)			CPX Z (2)	SBC Z (2)	INC Z (2)	
F	BEQ (2)	SBC (IND),Y (2)				SBC Z,X (2)	INC Z,X (2)	

Figure 5. Ce tableau indicateur des codes opération est la version condensée du tableau présenté dans l'appendice 1, à la fin de ce livre. L'information contenue dans une colonne et qui représente le quartet droit du code opération, joue un rôle important dans le déroulement du programme déterminant la longueur de l'instruction correspondante.

conséquent, ØØ, Ø1, Ø2, Ø3).

De loin, il nous semble vous entendre protester: C'est très bien, tout cela marche pour les 9 colonnes 1, 3, 5, 6, 7, 8, B, D et F, qui, chacune, se composent de 16 instructions de 0, 1, 2 ou 3 octets (dans la figure 5, après le mnémonique, la longueur de l'instruction est indiquée entre parenthèses), mais, les 7 colonnes restantes comportent des instructions de longueurs différentes! Dans ce cas, on ne peut se servir de LENTBL. Rassurez-vous, il existe une solution.

Le cas "difficile" des colonnes irrégulières

Pour commencer, nous négligerons les emplacements vides de toutes les colonnes ne se composant pas exclusivement d'emplacements vides. En faisant cela, nous ajoutons, en fait, aux neuf colonnes régulières initiales, les colonnes 2 (qui ne comporte que LDX IMM, et se trouve donc assimilée à une colonne abritant exclusivement des instructions de 2 octets), 4 (qui, à part les emplacements vides, ne comporte que des instructions de 2 octets), A (qui, en dehors des emplacements vides, ne comporte que des instructions de 1 octet), C (qui, hors les emplacements vides, regroupe exclusivement des instructions de 3 octets), et E (qui, sauf à l'emplacement vide 9E, présente des instructions à 3 octets). D'ailleurs, les emplacements vides pourront être tous visualisés séparément, ce que nous étudierons ultérieurement.

Restent les colonnes Ø et 9, dont nous n'avons encore rien dit. La colonne Ø contient principalement des instructions de 2 octets auxquelles s'ajoutent celles qui constituent des exceptions: BRK, RTI et RTS de chacune 1 octet, et JSR de 3 octets. Quant à la colonne 9, elle contient des instructions comportant certaines 2 octets, d'autres 3 octets.

En dépit de ces exceptions, il nous est toujours possible de faire usage du système de tableau indicateur déjà décrit. A condition que nous tenions compte de deux points: dans le tableau indicateur se trouvent des informations relatives à la longueur d'instruction *la plus représentée* dans la

Caractère hexadécimal droit

Caractère hexadécimal gauche	8	9	A	B	C	D	E	F	
Ø	PHP (1)	ORA # (2)	ASL A (1)			ORA ABS (3)	ASL ABS (3)		Ø
1	CLC (1)	ORA ABS,Y (3)				ORA ABS,X (3)	ASL ABS,X (3)		1
2	PLP (1)	AND # (2)	ROL A (1)		BIT ABS (3)	AND ABS (3)	ROL ABS (3)		2
3	SEC (1)	AND ABS,Y (3)				AND ABS,X (3)	ROL ABS,X (3)		3
4	PHA (1)	EOR # (2)	LSR A (1)		JMP ABS (3)	EOR ABS (3)	LSR ABS (3)		4
5	CLI (1)	EOR ABS,Y (3)				EOR ABS,X (3)	LSR ABS,X (3)		5
6	PLA (1)	ADC # (2)	ROR A (1)		JMP IND (3)	ADC ABS (3)	ROR ABS (3)		6
7	SEI (1)	ADC ABS,Y (3)				ADC ABS,X (3)	ROR ABS,X (3)		7
8	DEY (1)		TXA (1)		STY ABS (3)	STA ABS (3)	STX ABS (3)		8
9	TYA (1)	STA ABS,Y (3)	TXS (1)			STA ABS,X (3)			9
A	TAY (1)	LDA # (2)	TAX (1)		LDY ABS (3)	LDA ABS (3)	LDX ABS (3)		A
B	CLV (1)	LDA ABS,Y (3)	TSX (1)		LDY ABS,X (3)	LDA ABS,X (3)	LDX ABS,Y (3)		B
C	INY (1)	CMP # (2)	DEX (1)		CPY ABS (3)	CMP ABS (3)	DEC ABS (3)		C
D	CLD (1)	CMP ABS,Y (3)				CMP ABS,X (3)	DEC ABS,X (3)		D
E	INX (1)	SBC # (2)	NOP (1)		CPX ABS (3)	SBC ABS (3)	INC ABS (3)		E
F	SED (1)	SBC ABS,Y (3)				SBC ABS,X (3)	INC ABS,X (3)		F

colonne (donc, pour Y=ØØ X=Ø2, pour Y=Ø9 X=Ø3) *et* les exceptions minoritaires des colonnes Ø et 9 doivent être visualisées séparément. Ce dernier point se réduit à ce que les instructions de 1 octet BRK, RTI et RTS de la colonne Ø, les instructions 2 octets de la colonne Ø (JSR) et 3 octets de la colonne 9 soient visualisées à part, en premier lieu.

Les exceptions confirment la règle

Le sous-programme LENACC débute par le chargement de Ø1 dans X. Pourquoi Ø1? Parce que le microprocesseur commence par "filtrer" les trois instructions de 1 octet de la colonne Ø. Cela se fait avec trois combinaisons successivement répétées d'instructions CMP . . . BEQ, grâce auxquelle l'octet tapé au moyen des touches est comparé à tour de rôle au code opération de BRK, RTI et RTS. S'il correspond au code opération de l'une des trois instructions, le microprocesseur se branche sur le sous-programme LENEND et le continu de X=Ø1 est écrit dans BYTES.
Ensuite, Ø3 est chargé dans X et l'instruction de trois octets, JSR, de la colonne Ø est filtrée grâce à une instruction CMP # 2Ø suivie d'une instruction BEQ. Si l'octet tapé est identique au code opération de JSR, le microprocesseur saute au sous-programme LENEND et Ø3 est inscrit à l'emplacement mémoire BYTES.
Puis, à l'aide d'un test astucieux, les instructions de 3 octets et celles de 2 octets de la colonne 9 sont séparées. Pour cela, la fonction AND est appliquée avec 1F (masque logique) à l'octet introduit au moyen du jeu de touches et chargé dans l'accumulateur. Il importe de se remémorer la notation de la fonction AND:

$$1 \wedge X = X \text{ et } Ø \wedge X = X$$

Elle est appliquée bit par bit avec le contenu de l'accumulateur. Le résultat est ensuite rangé dans celui-ci:
contenu de l'accu avant l'opération logique : XXXXXXXX
fonction logique AND avec 1F : ØØØ11111
contenu de l'accu après l'opération logique : ØØØXXXXX
X pouvant avoir n'importe quelle valeur (don't care), 1 ou Ø.
Nous constatons qu'après l'utilisation du masque, les cinq bits de droite de l'octet n'ont pas été modifiés. Le quartet droit est d'ailleurs nécessaire, dans la suite du programme, à la détermination de la colonne à laquelle se rattache l'octet tapé (voir le tableau indicateur).

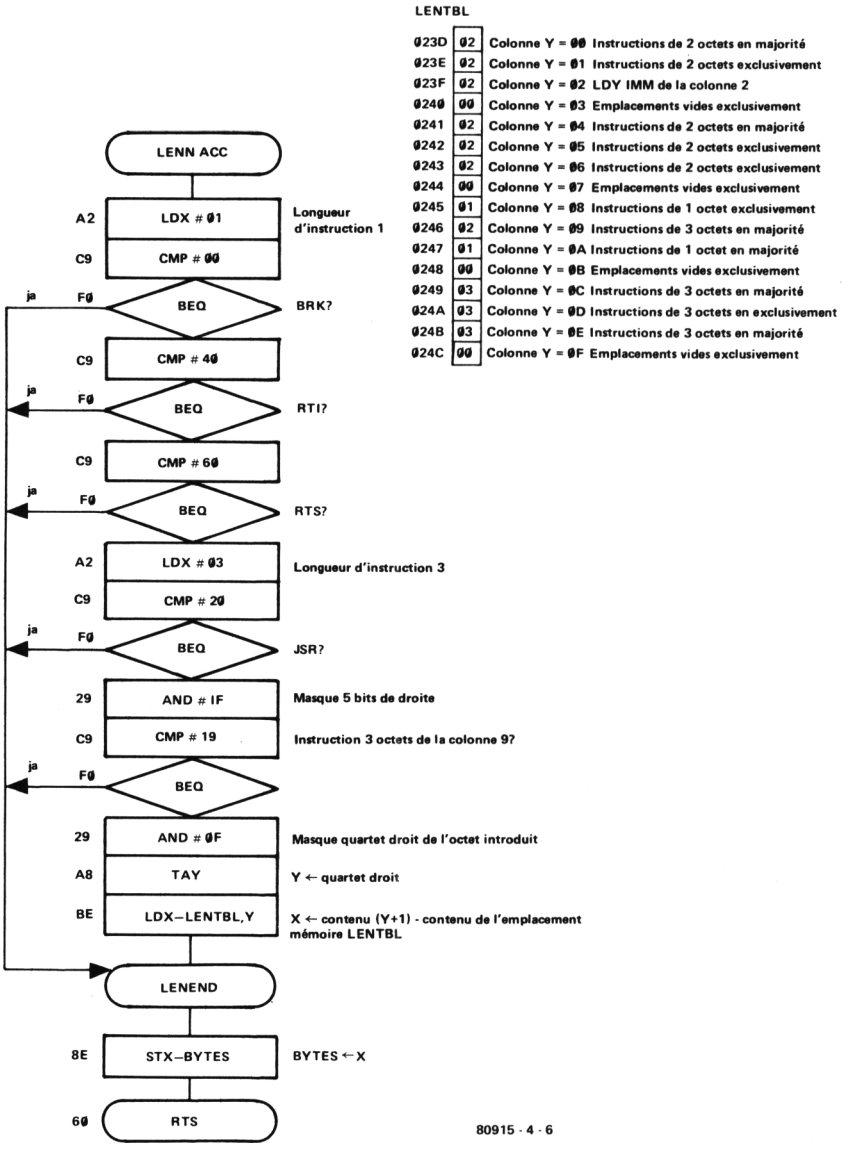


Figure 6. Le sous-programme LENACC assure que, après que deux touches numériques (0 . . . F) aient été pressées et que leurs valeurs aient été déposées dans l'accumulateur, le code opération ajouté au contenu de l'accumulateur contribue à la détermination de la longueur d'instruction convenable (01, 02 et 03). Si le quartet droit du contenu de l'accumulateur, c'est-à-dire la valeur de la seconde touche pressée, est égal à 3, 7, B ou F (ce qui correspond à l'une ces colonnes vides 3, 7, B ou F), la longueur d'instruction 00 est attribuée à l'octet introduit.

Après avoir vu comment se déroule le processus, examinons la justification de celui-ci. Elle concerne la succession des instructions de 2 et 3 octets de la colonne 9. Le quartet gauche d'une instruction de cette colonne est pair lorsqu'elle est de 2 octets et impair lorsqu'elle est de 3 octets. Dans les instructions de 3 octets de la colonne 9, les quatre bits de droite sont égaux à 9 (valeur hexadécimale); le quartet gauche étant impair, le quatrième bit de droite est un 1. Les trois bits de gauche ont été mis à Ø après utilisation du masque. En d'autres termes, la comparaison avec 19 (CMP IMM) permet de déterminer si l'instruction est de 3 octets ou non. Si c'est le cas (test suivi de l'instruction BEQ), le contenu de X (Ø3) est écrit dans BYTES (branchement vers LENEND).
C'est ainsi que tous les cas d'exception sont filtrés. Viennent ensuite l'opération logique AND (masque) avec ØF grâce à laquelle le quartet droit de l'octet introduit est isolé, le transfert du contenu de l'accumulateur dans le registre Y (TAY) et le chargement de la "ligne" correspondante du tableau (en fonction de Y) dans le registre X; lorsque le cas est "régulier", le branchement vers LENEND est opéré.

Identification de tous les emplacements vides

Avec le sous-programme LENACC de la figure 6, seules les instructions, dont l'octet est nul, des colonnes 3, 7, B et F sont à identifier en tant que telles (X=ØØ); les 41 restantes sont écartées. Bien que la figure 6 ne montre pas comment l'opération se déroule, nous allons indiquer ci-après le processus grâce auquel la totalité des 105 emplacements vides du tableau des codes opération de la figure 5 seront dépistés:
Le sous-programme LENACC débute par le transfert de ØØ dans le registre X. Au moyen de 26 combinaisons CMP IMM . . . BEQ, il est déterminé si l'octet introduit correspond à l'un des emplacements vides des colonnes Ø, 4, 9, A, C ou E. Dans l'affirmative, l'instruction BEQ concernée branche l'ordinateur vers LENEND; dans la négative, Ø2 est chargé dans X et, à l'aide d'une instruction CMP#A2 suivie d'une instruction BEQ, le programme détecte si le code opératoire est celui de l'instruction LDX IMM, la seule présente dans la colonne 2. A l'emplacement mémoire LENTBL, où est rangé le contenu du tableau indicateur, la valeur de Y est modifiée de Ø2 à ØØ. Ensuite, le programme de la figure 6 se poursuit sans changement par rapport au processus déjà cité.

Début et fin de programme

Les figures 7 et 8 donnent respectivement l'ordinogramme général et l'ordinogramme détaillé du programme complet du "calibre d'instruction". Que se passe-t-il, en somme? Nous pressons deux touches numériques (Ø . . . F) et l'octet correspondant est chargé dans l'accumulateur tandis que l'affichage le visualise sur les deux afficheurs de gauche. Les deux afficheurs centraux font apparaître la longueur de l'instruction correspondant à l'octet (code opération) en illuminant Ø1, Ø2 ou Ø3. Dans le cas où le quartet droit de l'octet est égal à 3, 7, B ouf F, c'est ØØ qui est affiché. Les deux afficheurs de droite font apparaître "FF" en permanence. En nous reportant à l'ordinogramme général de la figure 7, nous constatons qu'après START, FF est chagé dans les tampons d'affichage. Cette opération est suivie de la section de programme TWONI (l'expression

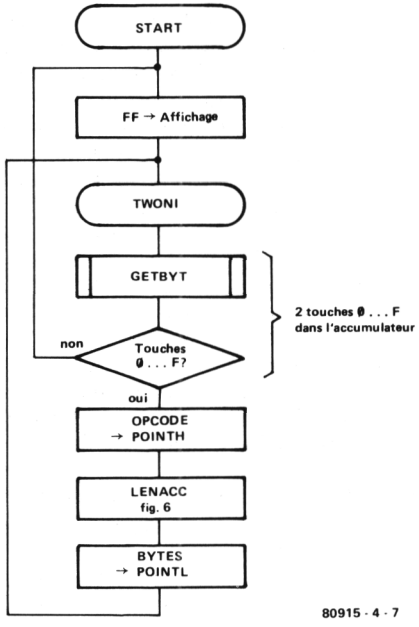


Figure 7. Ordinogramme général du programme "calibre d'instruction".

mnémonique américaine signifiant "TWO NIbbles" = deux quartets). La routine moniteur GETBYT garantit que la valeur hexadécimale des touches pressées consécutivement soit déposée dans l'accumulateur. Le quartet gauche du contenu de l'accumulateur traduit la valeur de la touche pressée la première, le quartet droit celle de la touche pressée ensuite.
Dans ce concept, nous laissons volontairement de côté la manipulation des touches de commande pour ne nous intéresser qu'à celle des touches numériques Ø . . . F. La routine GETBYT est alors suivie d'un test permettant de déterminer si les deux touches pressées sont celles de caractères hexadécimaux. A supposer que ce ne soit pas le cas, le programme parcourt une boucle d'attente, car les touches de commande sont ignorées et ce ne sera que lorsque deux touches numériques seront pressées que le microprocesseur procédera à la copie du contenu de l'accumulateur dans le tampon d'affichage POINTH. A partir de ce moment, les deux afficheurs de gauche font apparaître le code opération introduit et il est fail appel au sous-programme LENNAC. Celui-ci veille à ce que l'emplacement mémoire BYTES contienne la longueur de l'instruction. Puis, s'opère le transfert, par l'intermédiaire de l'accumulateur, du contenu de BYTES vers l'emplacement mémoire POINTL, grâce à quoi, après le branchement vers le sous-programme TWONI, la routine GETBYT visualise la longueur de l'instruction sur les deux afficheurs centraux. Car, nous le savons, imbriquée dans GETBYT, la routine SCANDS contribue à l'affichage du contenu des trois tampons des afficheurs.

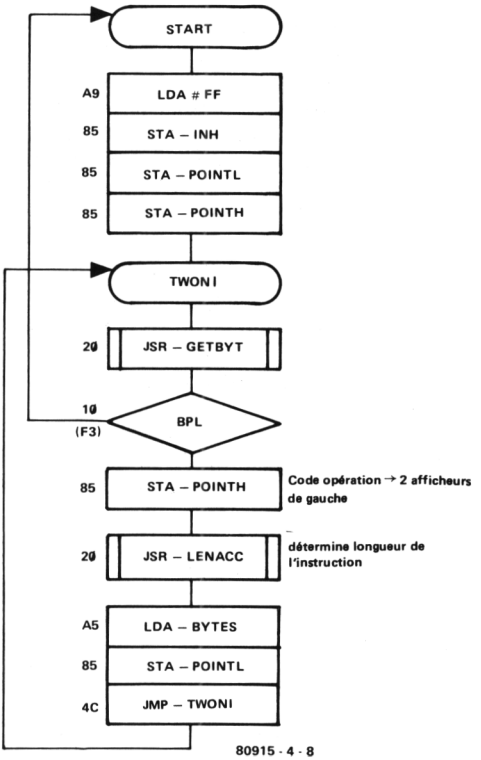


Figure 8. Ordinogramme détaillé du programme "calibre d'instruction".

L'ordinogramme de la figure 8 est une élaboration, sous forme d'instructions, de celui de la figure 7. Il faut encore préciser le rôle de l'instruction BPL, après GETBYT. Dans le cours du déroulement de celle-ci, une instruction CMP#1Ø permet de faire la distinction entre les touches de commande (valeur $\geqslant$ 1Ø) et numériques (ØØ . . . ØF). En ce qui concerne les touches dont la valeur est inférieure à 1Ø, la comparaison a pour conséquence la mise à 1 de l'indicateur (flag) N. Et l'instruction BPL signifie précisément branchement conditionnel si plus, en fonction de l'état de l'indicateur N. Il s'ensuit que le branchement est effectué si N=Ø, autrement dit, si la valeur de la touche est égale ou supérieure à 1Ø, ce qui signifie qu'il s'agit d'une touche de commande.
Nous en arrivons finalement à la séquence de frappe des touches de la figure 9, le programme se composant de 77 octets mobilisant 77 emplacements mémoire. L'adresse de début est ØØØØ et le programme principal (figure 8) se déroule de l'adresse ØØØØ à l'adresse Ø218 incluse, le sous-programme LENACC (figure 6) de l'adresse Ø219 à l'adresse Ø23C incluse, les 16 emplacements mémoire suivants de Ø23D à Ø24C inclus étant réservés au tableau indicateur LENTBL.

L'abondance des commentaires de la figure 9 laisse peu de choses à ajouter, sinon que l'emplacement mémoire BYTES se trouve sur la page zéro (adresse ØØEØ). Cela résulte du fait que le programme moniteur contient également une routine, permettant la mesure de la longueur de l'instruction (OPLEN, dont l'adresse de début est 1E5C), et dont le point d'aboutissement est BYTES (adresse ØØF6 et emplacements mémoire ØØE1 . . . ØØFF de la page zéro de la RAM) pour l'écriture de la longueur de l'instruction.

Un test à votre intention

Il a déjà été mentionné que les emplacements vides du tableau des codes opération de la figure 5 n'étaient pas tous reconnus comme correspondant à des instructions d'octet nul. Il a été également indiqué le moyen de remédier à cet inconvénient. C'est une excellente mise à l'épreuve de vos nouvelles aptitudes à la programmation que de traduire ces renseignements concrets en instructions et en une séquence de frappe de touches adéquates. Après que vous aurez introduit le programme à l'aide du clavier, le Junior Computer vous dira si votre travail a été bien fait.

	toetsen			adres	data	comments
RTS	AD			xxxx	xx	
Ø	2	Ø	Ø	Ø2ØØ	xx	adresse de début du calibre instruction
DA		A	9	Ø2ØØ	A9	LDA IMM [START]
+		F	F	Ø2Ø1	FF	FF → accumulateur
+		8	5	Ø2Ø2	85	STA Z
+		F	9	Ø2Ø3	F9	accu → INH (adresse ØØF9)
+		8	5	Ø2Ø4	85	STA Z
+		F	A	Ø2Ø5	FA	accu → POINTL (adresse ØØFA)
+		8	5	Ø2Ø6	85	STA Z
+		F	B	Ø2Ø7	FB	accu → POINTH (adresse ØØFB)
+		2	Ø	Ø2Ø8	2Ø	JSR- [TWONI]
+		6	F	Ø2Ø9	6F	ADL } de GETBYT (moniteur;
+		1	D	Ø2ØA	1D	ADH } adresse 1D6F)
+		1	Ø	Ø2ØB	1Ø	BPL 2 touches Ø . . . F pressées?
+		F	3	Ø2ØC	F3	sinon, F3 emplacements vers l'arrière (= START)
+		8	5	Ø2ØD	85	STA Z (octet dans l'accu)
+		F	B	Ø2ØE	FB	accu (= code opération) → POINTH (adresse ØØFB)
+		2	Ø	Ø2ØF	2Ø	JSR-
+		1	9	Ø21Ø	19	ADL } de LENACC
+		Ø	2	Ø211	Ø2	ADH }
+		A	5	Ø212	A5	LDA Z
+		E	Ø	Ø213	EØ	BYTES (adresse ØØEØ) → accu
+		8	5	Ø214	85	STA Z

+	F	A	Ø215	FA	accu → POINTL (adresse ØØFA)
+	4	C	Ø216	4C	JMP ABS
+	Ø	8	Ø217	Ø8	ADL } de l'adresse du saut
+	Ø	2	Ø218	Ø2	ADH } (TWONI)
+	A	2	Ø219	A2	LDX IMM; sous-programme LENACC
+	Ø	1	Ø21A	Ø1	Ø1 → X; longueur d'instruction 1
+	C	9	Ø21B	C9	CMP IMM
+	Ø	Ø	Ø21C	ØØ	avec ØØ
+	F	Ø	Ø21D	FØ	BEQ BRK?
+	1	A	Ø21E	1A	LENEND est à 1A emplacements en avant
+	C	9	Ø21F	C9	CMP IMM
+	4	Ø	Ø22Ø	4Ø	avec 4Ø
+	F	Ø	Ø221	FØ	BEQ RTI?
+	1	6	Ø222	16	LENEND est à 16 emplacements en avant
+	C	9	Ø223	C9	CMP IMM
+	6	Ø	Ø224	6Ø	avec 6Ø
+	F	Ø	Ø225	FØ	BEQ RTS?
+	1	2	Ø226	12	LENEND est à 12 emplacements en avant
+	A	2	Ø227	A2	LDX IMM
+	Ø	3	Ø228	Ø3	Ø3 → X; longueur d'instruction 3
+	C	9	Ø229	C9	CMP IMM
+	2	Ø	Ø22A	2Ø	avec 2Ø
+	F	Ø	Ø22B	FØ	BEQ JSR?
+	Ø	C	Ø22C	ØC	LENEND est à ØC emplacements en avant
+	2	9	Ø22D	29	AND IMM
+	1	F	Ø22E	1F	avec 1F; masque 5 bits de droite
+	C	9	Ø22F	C9	CMP IMM
+	1	9	Ø23Ø	19	avec 19
+	F	Ø	Ø231	FØ	BEQ; instruction 3 octets de la colonne 9?
+	Ø	6	Ø232	Ø6	LENEND est à Ø6 emplacements en avant
+	2	9	Ø233	29	AND IMM
+	Ø	F	Ø234	ØF	avec ØF; masque quartet droit
+	A	8	Ø235	A8	TAY; quartet droit → Y
+	B	E	Ø236	BE	LDX ABS,Y; (Y + 1) - contenu de l'emplacement mémoire LENTBL
+	3	D	Ø237	3D	ADL } de LENTBL
+	Ø	2	Ø238	Ø2	ADH } (tableau indicateur)
+	8	E	Ø239	8E	STX ABS LENEND
+	E	Ø	Ø23A	EØ	ADL de BYTES
+	Ø	Ø	Ø23B	ØØ	ADH
+	6	Ø	Ø23C	6Ø	RTS retour vers le programme principal
+	Ø	2	Ø23D	Ø2	colonne Ø; Y = ØØ LENTBL

+		Ø	2	Ø23E	Ø2	colonne 1 ; Y = Ø1
+		Ø	2	Ø23F	Ø2	colonne 2 ; Y = Ø2
+		Ø	Ø	Ø24Ø	ØØ	colonne 3 ; Y = Ø3
+		Ø	2	Ø241	Ø2	colonne 4 ; Y = Ø4
+		Ø	2	Ø242	Ø2	colonne 5 ; Y = Ø5
+		Ø	2	Ø243	Ø2	colonne 6 ; Y = Ø6
+		Ø	Ø	Ø244	ØØ	colonne 7 ; Y = Ø7
+		Ø	1	Ø245	Ø1	colonne 8 ; Y = Ø8
+		Ø	2	Ø246	Ø3	colonne 9 ; Y = Ø9
+		Ø	1	Ø247	Ø1	colonne A ; Y = ØA
+		Ø	Ø	Ø248	ØØ	colonne B ; Y = ØB
+		Ø	3	Ø249	Ø3	colonne C ; Y = ØC
+		Ø	3	Ø24A	Ø3	colonne D ; Y = ØD
+		Ø	3	Ø24B	Ø3	colonne E ; Y = ØE
+		Ø	Ø	Ø24C	ØØ	colonne F ; Y = ØF
AD						
Ø	2	Ø	Ø			adresse de debut
GO						debut du programme
		A	9	A9Ø2	FF	
		Ø	3	Ø3ØØ	FF	
		D	2	D2Ø2	FF	
		9	E	9EØ3	FF	
		D	5	D5Ø2	FF	
		etc . . .				

Figure 9. Séquence d'introduction du programme des figures 6 et 8, grâce au jeu de touches du clavier du Junior Computer.

Ainsi s'achèvent ce chapitre 4 et le premier livre consacré au Junior Computer; nous espérons bien vous retrouver avec le second ouvrage, Junior Computer 2. Vous y connaîtrez tout un choix de programmes qui vous feront encore mieux goûter la joie de la découverte et de l'utilisation du microordinateur.

Appendice 1

Tableau des codes opération dans l'ordre de notation hexadécimale 00-FF.
Les combinaisons de caractères hexadécimaux non utilisées sont également reproduites.

00	BRK	20	JSR ABS	40	RTI IMP	60	RTS IMP
01	ORA (IND,X)	21	AND (IND,X)	41	EOR (IND,X)	61	ADC (IND,X)
02	–	22	–	42	–	62	–
03	–	23	–	43	–	63	–
04	–	24	BIT Z	44	–	64	–
05	ORA Z	25	AND Z	45	EOR Z	65	ADC Z
06	ASL Z	26	ROL Z	46	LSR Z	66	ROR Z
07	–	27	–	47	–	67	–
08	PHP IMP	28	PLP IMP	48	PHA IMP	68	PLA IMP
09	ORA IMM	29	AND IMM	49	EOR IMM	69	ADC IMM
0A	ASL A	2A	ROL A	4A	LSR A	6A	ROR A
0B	–	2B	–	4B	–	6B	–
0C	–	2C	BIT ABS	4C	JMP ABS	6C	JMP IND
0D	ORA ABS	2D	AND ABS	4D	EOR ABS	6D	ADC ABS
0E	ASL ABS	2E	ROL ABS	4E	LSR ABS	6E	ROR ABS
0F	–	2F	–	4F	–	6F	–
10	BPL REL	30	BMI REL	50	BVC REL	70	BVS REL
11	ORA (IND), Y	31	AND (IND), Y	51	EOR (IND), Y	71	ADC (IND),Y
12	–	32	–	52	–	72	–
13	–	33	–	53	–	73	–
14	–	34	–	54	–	74	–
15	ORA Z,X	35	AND Z,X	55	EOR Z,X	75	ADC Z,X
16	ASL Z,X	36	ROL Z,X	56	LSR Z,X	76	ROR Z,X
17	–	37	–	57	–	77	–
18	CLC IMP	38	SEC IMP	58	CLI IMP	78	SEI IMP
19	ORA ABS,Y	39	AND ABS,Y	59	EOR ABS,Y	79	ADC ABS,Y
1A	–	3A	–	5A	–	7A	–
1B	–	3B	–	5B	–	7B	–
1C	–	3C	–	5C	–	7C	–
1D	ORA ABS,X	3D	AND ABS,X	5D	EOR ABS,X	7D	ADC ABS,X
1E	ASL ABS,X	3E	ROL ABS,X	5E	LSR ABS,X	7E	ROR ABS,X
1F	–	3F	–	5F	–	7F	–

80	–	A0	LDY IMM	C0	CPY IMM	E0	CPX IMM
81	STA (IND,X)	A1	LDA (IND,X)	C1	CMP (IND,X)	E1	SBC (IND,X))
82	–	A2	LDX IMM	C2	–	E2	–
83	–	A3	–	C3	–	E3	–
84	STY Z	A4	LDY Z	C4	CPY Z	E4	CPX Z
85	STA Z	A5	LDA Z	C5	CMP Z	E5	SBC Z
86	STX Z	A6	LDX Z	C6	DEC Z	E6	INC Z
87	–	A7	–	C7	–	E7	–
88	DEY IMP	A8	TAY IMP	C8	INY IMP	E8	INX IMP
89	–	A9	LDA IMM	C9	CMP IMM	E9	SBC IMM
8A	TXA IMP	AA	TAX IMP	CA	DEX IMP	EA	NOP IMP
8B	–	AB	–	CB	–	EB	–
8C	STY ABS	AC	LDY ABS	CC	CPY ABS	EC	CPX ABS
8D	STA ABS	AD	LDA ABS	CD	CMP ABS	ED	SBC ABS
8E	STX ABS	AE	LDX ABS	CE	DEC ABS	EE	INC ABS
8F	–	AF	–	CF	–	EF	–
90	BCC REL	B0	BCS REL	D0	BNE REL	F0	BEQ REL
91	STA (IND),Y	B1	LDA (IND),Y	D1	CMP (IND),Y	F1	SBC (IND),Y
92	–	B2	–	D2	–	F2	–
93	–	B3	–	D3	–	F3	–
94	STY Z,X	B4	LDY Z,X	D4	–	F4	–
95	STA Z,X	B5	LDA Z,X	D5	CMP Z,X	F5	SBC Z,X
96	STX Z,Y	B6	LDX Z,Y	D6	DEC Z,X	F6	INC Z,X
97	–	B7	–	D7	–	F7	–
98	TYA IMP	B8	CLV IMP	D8	CLD IMP	F8	SED IMP
99	STA ABS,Y	B9	LDA ABS,Y	D9	CMP ABS,Y	F9	SBC ABS,Y
9A	TXS IMP	BA	TSX IMP	DA	–	FA	–
9B	–	BB	–	DB	–	FB	–
9C	–	BC	LDY ABS,X	DC	–	FC	–
9D	STA ABS,X	BD	LDA ABS,X	DD	CMP ABS,X	FD	SBC ABS,X
9E	–	BE	LDX ABS,Y	DE	DEC ABS,X	FE	INC ABS,X
9F	–	BF	–	DF	–	FF	–

Commentaires. Le code opération est suivi immédiatement de l'abréviation anglo-saxonne (mnémonique) de l'instruction. Celle-ci se compose de trois majuscules. Lorsqu'une instruction comporte plusieurs modes d'adressage, les trois lettres capitales sont suivies d'une indication du mode concerné par l'opération. Voici les diverses significations:

IMM	Adressage immédiat (immediate addressing)
ABS	Adressage absolu (absolute addressing)
Z	Adressage page-zéro (zero page addressing)
A	Adressage de l'accumulateur (Accumulator addressing)
(IND,X)	Adressage indirect indexé X (indexed indirect addressing)
(IND,Y)	Adressage indexé Y indirect (indirect indexed addressing)
Z,X	Adressage indexé X en page-zéro (zero page indexed, X addressing)
Z,Y	Adressage indexé Y en page-zéro (zero page indexed, Y addressing)
ABS,X	Adressage indexé X absolu (absolute indexed, X addressing)
ABS,Y	Adressage indexé Y absolu (absolute indexed, Y addressing)
IND	Adressage indirect (indirect addressing)

Appendice 2

Tableau des 56 instructions du microprocesseur 6502 disposées selon un ordre alphabétique. Un grand nombre d'instructions autorisent plusieurs modes d'adressage conduisant à un total de 149 codes opération différents.

Abréviation mnémonique et définition	Mode d'adressage	Code opération hexadécimal	Nombre d'impulsions d'horloge (N)	Nombre d'octets	Indicateur(s) affecté(s)
ADC Addition de la mémoire à l'accumulateur avec retenue A+M+C → A (1) (4)	IMM ABS Z (IND,X) (IND,Y) Z.X ABS,X ABS,Y	69 6D 65 61 71 75 7D 79	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	NV----ZC
AND Fonction ET logique appliquée à la mémoire avec l'accumulateur A ∧ M → A (1)	IMM ABS Z (IND,X) (IND,Y) Z,X ABS,X ABS,Y	29 2D 25 21 31 35 3D 39	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N-----Z-
ASL Décalage arithmétique à gauche d'un bit (accu ou mémoire) C ← 7 ... 0 ← 0	ABS Z A Z,X ABS,X	0E 06 0A 16 1E	6 5 2 6 7	3 2 1 2 3	N-----ZC
BCC Branchement si pas de retenue (2) Branchement si C = 0	REL	90	2	2	--------
BCS Branchement si retenue (2) Branchement si C = 1	REL	B0	2	2	--------
BEQ Branchement si égalité (si Z = 1, c'est-à-dire, si résultat = 0) (2)	REL	F0	2	2	--------

Abréviation mnémonique et définition	Mode d'adressage	Code opération hexadécimal	Nombre d'impulsions d'horloge (N)	Nombre d'octets	Indicateur(s) affecté(s)
BIT Test de bits en mémoire: A $\wedge$ M $M_7 \rightarrow N$; $N_6 \rightarrow V$	ABS Z	2C 24	4 3	3 2	M_7M_6––Z–
BMI Branchement à moins (2) Branchement si N = 1 (résultat $<$ Ø)	REL	3Ø	2	2	––––––––
BNE Branchement si non égal à zéro (2) Branchement si Z = Ø	REL	DØ	2	2	––––––––
BPL Branchement si plus (2) Branchement si N = Ø	REL	1Ø	2	2	––––––––
BRK Interruption software (logiciel)	IMP	ØØ	7	1	––––1–1–– B I
BVC Branchement si pas de débordement (2) Branchement si V = Ø	REL	5Ø	2	2	––––––––
BVS Branchement si débordement (2) Branchement si V = 1	REL	7Ø	2	2	––––––––
CLC Mise à zéro de la retenue Ø $\rightarrow$C	IMP	18	2	1	––––––––Ø C
CLD Mise à zéro du mode décimal Ø $\rightarrow$ D	IMP	D8	2	1	–––––Ø––– D

Abréviation mnémonique et définition	Mode d'adressage	Code opération hexadécimal	Nombre d'impulsions d'horloge (N)	Nombre d'octets	Indicateur(s) affecté(s)
CLI Mise à zéro du masque des interruptions Ø → I	IMP	58	2	1	------Ø-- I
CLV Mise à zéro de l'indicateur de débordement Ø → V	IMP	B8	2	1	Ø------- V
CMP Comparer mémoire à l'accumulateur A–M	IMM ABS Z (IND,X) (IND,Y) Z,X ABS,X ABS,Y	C9 CD C5 C1 D1 D5 DD D9	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N-----ZC
CPX Comparer mémoire à index X X–M	IMM ABS Z	EØ EC E4	2 4 3	2 3 2	N-----ZC
CPY Compare mémoire à index Y M–Y	IMM ABS Z	CØ CC C4	2 4 3	2 3 2	N-----ZC
DEC Décrémenter de 1 la mémoire M–1 → M	ABS Z Z,X ABS,X	CE C6 D6 DE	6 5 6 7	3 2 2 3	N-----Z-
DEX Décrémenter de 1 l'index X X–1 → X	IMP	CA	2	1	N-----Z-
DEY Décrémenter de 1 l'index Y Y–1 → Y	IMP	88	2	1	N-----Z-

Abréviation mnémonique et définition	Mode d'adressage	Code opération hexadécimal	Nombre d'impulsions d'horloge (N)	Nombre d'octets	Indicateur(s) affecté(s)
EOR Ou exclusif avec l'accumulateur A ∀ M → A (1)	IMM ABS Z (IND,X) (IND,Y) Z,X ABS,X ABS,Y	49 4D 45 41 51 55 5D 59	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N–––––Z–
INC Incrémenter en mémoire M + 1 →M	ABS Z Z,X ABS,X	EE E6 F6 FE	6 5 6 7	3 2 2 3	N–––––Z–
INX Incrémenter index X X + 1 → X	IMP	E8	2	1	N–––––Z–
INY Incrémenter index Y Y + 1 →Y	IMP	C8	2	1	N–––––Z–
JMP Saut à une adresse (PC + 1) →PCL (PC + 2) →PCH	ABS IND	4C 6C	3 5	3 3	––––––––
JSR Saut à un sous-programme avec sauvegarde de l'adresse de retour PC + 2 ↓ (PC + 1) → PCL (PC + 2) → PCH	ABS	20	6	3	––––––––
LDA Chargement de l'accumulateur avec la mémoire M → A (1)	IMM ABS Z (IND,X) (IND,Y) Z,X ABS,X ABS,Y	A9 AD A5 A1 B1 B5 BD B9	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N–––––Z–
LDX Chargement de l'index X avec la mémoire M → X (1)	IMM ABS Z Z,Y ABS,Y	A2 AE A6 B6 BE	2 4 3 4 4	2 3 2 2 3	N–––––Z–

Abréviation mnémonique et définition	Mode d'adressage	Code opération hexadécimal	Nombre d'impulsions d'horloge (N)	Nombre d'octets	Indicateur(s) affecté(s)
LDY Chargement de l'index Y avec la mémoire M → Y (1)	IMM ABS Z Z,X ABS,X	A0 AC A4 B4 BC	2 4 3 4 4	2 3 2 2 3	N————Z–
LSR Décalage logique à droite (mémoire ou accumulateur) 0 →7 0 →C	ABS Z A Z,X ABS,X	4E 46 4A 56 5E	6 5 2 6 7	3 2 1 2 3	0————ZC N
NOP Pas d'opération	IMP	EA	2	1	————————
ORA Ou inclusif avec l'accumulateur A V M → A	IMM ABS Z (IND,X) (IND,Y) Z,X ABS,X ABS,Y	09 0D 05 01 11 15 1D 19	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N————Z–
PHA Empiler le contenu de l'accumulateur A ↓	IMP	48	3	1	————————
PHP Empiler le contenu du registre P P ↓	IMP	08	3	1	————————
PLA Dépiler le contenu de l'accumulateur A ↑	IMP	68	4	1	N————Z–
PLP Dépiler le contenu du registre P P ↑	IMP	28	4	1	(restauré)
ROL Rotation à gauche (mémoire ou accu) 7 0 ← C	ABS Z Z,X ABS,X A	2E 26 36 3E 2A	6 5 6 7 2	3 2 2 3 1	N————ZC

Abréviation mnémonique et définition	Mode d'adressage	Code opération hexadécimal	Nombre d'impulsions d'horloge (N)	Nombre d'octets	Indicateur(s) affecté(s)
ROR Rotation à droite (mémoire ou accu) C → 7 ∅	ABS Z A Z,X ABS,X	6E 66 6A 76 7E	6 5 2 6 7	3 2 1 2 3	N——————ZC
RTI Retour d'interruption PC ↑ ; P ↑	IMP	4∅	6	1	(restauré)
RTS Retour de sous-programme PC ↑ ; PC+1 → PC	IMP	6∅	6	1	————————
SBC Soustraction avec retenue (emprunt) (3) A-M-$\overline{C}$ → A (1)	IMM ABS Z (IND,X) (IND,Y) Z,X ABS,X ABS,Y	E9 ED E5 E1 F1 F5 FD F9	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N——————ZC
SEC Mise à 1 de l'indicateur Carry 1 → C	IMP	38	2	1	————————1
SED Mise en mode décimal	IMP	F8	2	1	————1——— D
SEI Inhiber les interruptions: 1 → I	IMP	78	2	1	——————1—— I
STA Rangement de l'accumulateur en mémoire A → M	ABS Z (IND,X) ((ND,Y) Z,X ABS,X ABS,Y	8D 85 81 91 95 9D 99	4 3 6 6 4 5 5	3 2 2 2 2 3 3	————————
STX Ranger l'index X dans la mémoire X → M	ABS Z Z,Y	8E 86 96	4 3 4	3 2 2	————————

Abréviation mnémonique et définition	Mode d'adressage	Code opération hexadécimal	Nombre d'impulsions d'horloge (N)	Nombre d'octets	Indicateur(s) affecté(s)
STY Ranger l'index Y en mémoire Y → M	ABS Z Z,X	8C 84 94	4 3 4	3 2 2	————————
TAX Transfert de l'accu dans X A → X	IMP	AA	2	1	N————Z—
TAY Transfert de l'accu dans Y A → Y	IMP	A8	2	1	N————Z—
TSX Transfert de S dans X S → X	IMP	BA	2	1	N————Z—
TXA Transfert de X dans l'accu X → A	IMP	8A	2	1	N————Z—
TXS Transfert de X dans le registre S X →S	IMP	9A	2	1	N————Z—
TYA Transfert de Y dans l'accu Y → A	IMP	98	2	1	N————Z—

Remarques

(1) Plus 1 cycle (N+1) si une limite de page est franchie

(2) Plus 1 cycle (N+1) si le branchement a lieu
Plus 2 cycles (N+2) si l'on change de page

(3) Emprunt = borrow = non-Carry ($\overline{C}$)

(4) Dans les calculs décimaux, l'état de l'indicateur Z ne peut être utilisé
Il est nécessaire de tester l'accumulateur d'une autre manière pour le résultat nul

IMM	Adressage immédiat
ABS	Adressage absolu (direct)
Z	Adressage en page-zéro
A	Adressage accumulateur
IMP	Adressage implicite
(IND,X)	Adressage indirect indexé X
(IND,Y)	Adressage indexé Y indirect
Z,X	Adressage indexé X en page-zéro
Z,Y	Adressage indexé Y en page-zéro
ABS,X	Adressage indexé X absolu
ABS,Y	Adressage indexé Y absolu
REL	Adressage relatif
IND	Adressage indirect

Appendice 3

Listing en hexadécimal du programme moniteur du Junior Computer

Voici un aperçu du contenu des 1024 emplacements mémoire de l'EPROM IC2, sous forme hexadécimale. Ils sont disposés dans un tableau de 64 lignes de chacune 16 octets. La colonne de gauche donne l'adresse du premier octet de chaque ligne, les autres suivant dans l'ordre normal 1CØØ . . . 1CØF, puis 1C1Ø . . . 1C1F etc... Les numéros de colonne sont disposés à la partie supérieure (Ø . . . F).

Programme moniteur du Junior Computer

	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1CØØ:	85	F3	68	85	F1	68	85	EF	85	FA	68	85	FØ	85	FB	84
1C1Ø:	F4	86	F5	BA	86	F2	A2	Ø1	86	FF	4C	33	1C	A9	1E	8D
1C2Ø:	83	1A	A9	Ø4	85	F1	A9	Ø3	85	FF	85	F6	A2	FF	9A	86
1C3Ø:	F2	D8	78	2Ø	88	1D	DØ	FB	2Ø	88	1D	FØ	FB	2Ø	88	1D
1C4Ø:	FØ	F6	2Ø	F9	1D	C9	13	DØ	13	A6	F2	9A	A5	FB	48	A5
1C5Ø:	FA	48	A5	F1	48	A6	F5	A4	F4	A5	F3	4Ø	C9	1Ø	DØ	Ø6
1C6Ø:	A9	Ø3	85	FF	DØ	14	C9	11	DØ	Ø6	A9	ØØ	85	FF	FØ	ØA
1C7Ø:	C9	12	DØ	Ø9	E6	FA	DØ	Ø2	E6	FB	4C	33	1C	C9	14	DØ
1C8Ø:	ØB	A5	EF	85	FA	A5	FØ	85	FB	4C	7A	1C	C9	15	1Ø	EA
1C9Ø:	85	E1	A4	FF	DØ	ØD	B1	FA	ØA	ØA	ØA	ØA	Ø5	E1	91	FA
1CAØ:	4C	7A	1C	A2	Ø4	Ø6	FA	26	FB	CA	DØ	F9	A5	FA	Ø5	E1
1CBØ:	85	FA	4C	7A	1C	2Ø	D3	1E	A4	E3	A6	E2	E8	DØ	Ø1	C8
1CCØ:	86	E8	84	E9	A9	77	AØ	ØØ	91	E6	2Ø	4D	1D	C9	14	DØ
1CDØ:	2A	2Ø	6F	1D	1Ø	F7	85	FB	2Ø	6F	1D	1Ø	FØ	85	FA	2Ø
1CEØ:	D3	1E	AØ	ØØ	B1	E6	C5	FB	DØ	Ø7	C8	B1	E6	C5	FA	FØ
1CFØ:	D9	2Ø	5C	1E	2Ø	F8	1E	3Ø	E9	1Ø	3E	C9	1Ø	DØ	ØA	2Ø
1DØØ:	2Ø	1E	1Ø	C9	2Ø	47	1E	FØ	C1	C9	13	DØ	14	2Ø	2Ø	1E
1D1Ø:	1Ø	BB	2Ø	5C	1E	2Ø	F8	1E	A5	FD	85	F6	2Ø	47	1E	FØ
1D2Ø:	A9	C9	12	DØ	Ø7	2Ø	F8	1E	3Ø	AØ	1Ø	ØD	C9	11	DØ	Ø9
1D3Ø:	2Ø	83	1E	2Ø	EA	1E	4C	CA	1C	A9	EE	85	FB	85	FA	85
1D4Ø:	F9	A9	Ø3	85	F6	2Ø	8E	1D	DØ	FB	4C	CA	1C	A2	Ø2	AØ
1D5Ø:	ØØ	B1	E6	95	F9	C8	CA	1Ø	F8	2Ø	5C	1E	2Ø	8E	1D	DØ
1D6Ø:	FB	2Ø	8E	1D	FØ	FB	2Ø	8E	1D	FØ	F6	2Ø	F9	1D	6Ø	2Ø
1D7Ø:	5C	1D	C9	1Ø	1Ø	11	ØA	ØA	ØA	ØA	85	FE	2Ø	5C	1D	C9
1D8Ø:	1Ø	1Ø	Ø4	Ø5	FE	A2	FF	6Ø	AØ	ØØ	B1	FA	85	F9	A9	7F
1D9Ø:	8D	81	1A	A2	Ø8	A4	F6	A5	FB	2Ø	CC	1D	88	FØ	ØD	A5
1DAØ:	FA	2Ø	CC	1D	88	FØ	Ø5	A5	F9	2Ø	CC	1D	A9	ØØ	8D	81
1DBØ:	1A	AØ	Ø3	A2	ØØ	A9	FF	8E	82	1A	E8	E8	2D	8Ø	1A	88
1DCØ:	DØ	F5	AØ	Ø6	8C	82	1A	Ø9	8Ø	49	FF	6Ø	48	84	FC	4A

```
      Ø  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
1DDØ: 4A 4A 4A 2Ø DF 1D 68 29 ØF 2Ø DF 1D A4 FC 6Ø A8
1DEØ: B9 ØF 1F 8D 8Ø 1A 8E 82 1A AØ 7F 88 1Ø FD 8C 8Ø
1DFØ: 1A AØ Ø6 8C 82 1A E8 E8 6Ø A2 21 AØ Ø1 2Ø B5 1D
1EØØ: DØ Ø7 EØ 27 DØ F5 A9 15 6Ø AØ FF ØA BØ Ø3 C8 1Ø
1E1Ø: FA 8A 29 ØF 4A AA 98 1Ø Ø3 18 69 Ø7 CA DØ FA 6Ø
1E2Ø: 2Ø 6F 1D 1Ø 21 85 FB 2Ø 6Ø 1E 84 F7 84 FD C6 F7
1E3Ø: FØ 12 2Ø 6F 1D 1Ø ØF 85 FA C6 F7 FØ Ø7 2Ø 6F 1D
1E4Ø: 1Ø Ø4 85 F9 A2 FF 6Ø 2Ø A6 1E 2Ø DC 1E A2 Ø2 AØ
1E5Ø: ØØ B5 F9 91 E6 CA C8 C4 F6 DØ F6 6Ø AØ ØØ B1 E6
1E6Ø: AØ Ø1 C9 ØØ FØ 1A C9 4Ø FØ 16 C9 6Ø FØ 12 AØ Ø3
1E7Ø: C9 2Ø FØ ØC 29 1F C9 19 FØ Ø6 29 ØF AA BC 1F 1F
1E8Ø: 84 F6 6Ø A5 E6 85 EA A5 E7 85 EB A4 F6 B1 EA AØ
1E9Ø: ØØ 91 EA E6 EA DØ Ø2 E6 EB A5 EA C5 E8 DØ EC A5
1EAØ: EB C5 E9 DØ E6 6Ø A5 E8 85 EA A5 E9 85 EB AØ ØØ
1EBØ: B1 EA A4 F6 91 EA A5 EA C5 E6 DØ Ø6 A5 EB C5 E7
1ECØ: FØ 1Ø 38 A5 EA E9 Ø1 85 EA A5 EB E9 ØØ 85 EB 4C
1EDØ: AE 1E 6Ø A5 E2 85 E6 A5 E3 85 E7 6Ø 18 A5 E8 65
1EEØ: F6 85 E8 A5 E9 69 ØØ 85 E9 6Ø 38 A5 E8 E5 F6 85
1EFØ: E8 A5 E9 E9 ØØ 85 E9 6Ø 18 A5 E6 65 F6 85 E6 A5
1F ØØ: E7 69 ØØ 85 E7 38 A5 E6 E5 E8 A5 E7 E5 E9 6Ø 4Ø
1F1Ø: 79 24 3Ø 19 12 Ø2 78 ØØ 1Ø Ø8 Ø3 46 21 Ø6 ØE Ø2
1F2Ø: Ø2 Ø2 Ø1 Ø2 Ø2 Ø2 Ø1 Ø1 Ø2 Ø1 Ø1 Ø3 Ø3 Ø3 Ø3 6C
1F3Ø: 7A 1A 6C 7E 1A B1 E6 AØ FF C4 EE FØ ØD D1 EC DØ
1F4Ø: ØA 88 B1 EC AA 88 B1 EC AØ Ø1 6Ø 88 88 88 DØ E9
1F5Ø: 6Ø 38 A5 E4 E9 FF 85 EC A5 E5 E9 ØØ 85 ED A9 FF
1F6Ø: 85 EE 2Ø D3 1E 2Ø 5C 1E AØ ØØ B1 E6 C9 FF DØ 1D
1F7Ø: C8 B1 E6 A4 EE 91 EC 88 A5 E7 91 EC 88 A5 E6 91
1F8Ø: EC 88 84 EE 2Ø 83 1E 2Ø EA 1E 4C 65 1F 2Ø F8 1E
1F9Ø: 3Ø D3 2Ø D3 1E 2Ø 5C 1E AØ ØØ B1 E6 C9 4C FØ 16
1FAØ: C9 2Ø FØ 12 29 1F C9 1Ø FØ 1A 2Ø F8 1E 3Ø E6 A9
1FBØ: Ø3 85 F6 4C 33 1C C8 2Ø 35 1F FØ EE 91 E6 8A C8
1FCØ: 91 E6 DØ E6 C8 2Ø 35 1F FØ EØ 38 E5 E6 38 E9 Ø2
1FDØ: 91 E6 4C AA 1F D8 A9 ØØ 85 FB 85 FA 85 F9 2Ø 6F
1FEØ: 1D 1Ø F2 85 FB 2Ø 6F 1D 1Ø EB 85 FA 18 A5 FA E5
1FFØ: FB 85 F9 C6 F9 4C DE 1F FF FF 2F 1F 1D 1C 32 1F
```

Pour beaucoup de gens, "ordinateur" est synonyme de "moins de travail – plus de loisirs". En fait, il serait plus juste de penser: plus de travail dans le même temps. Le nombre, sans cesse croissant, d'adeptes du micro-ordinateur, suggère quant à lui la formule suivante: plus de travail pendant les loisirs.

Mais avant de s'y mettre, il faut en choisir un. Et devant la quantité d'offres de revues et de livres différents, de micros à construire soi-même ou prêts à l'emploi, plus d'un néophyte s'est senti à ce point désarçonné qu'il n'a même plus cherché à donner suite à ses velléités.

En réponse à ce besoin, Elektor a conçu le Junior Computer qui vient en aide à tous ceux pour qui l'arbre cache la forêt.

- Sur une carte unique, un micro-ordinateur performant mais économique,
- accompagné d'un véritable cours en 4 livres,
- de conception évolutive, comme son nom l'indique, puisque toutes les extensions, aussi bien d'usage général que d'usage spécifique, sont possibles.

ISBN 2-86661-001-6